

Doxxygen_MPDASTAR_DB Reference Manual

Generated by Doxygen 1.3.7

Fri Aug 25 11:14:40 2006

Contents

1	Doxygen_MPDS_DA_RT_SDB Hierarchical Index	1
1.1	Doxygen_MPDS _D A _R T _S DB Class Hierarchy	1
2	Doxygen_MPDS_DA_RT_SDB Class Index	3
2.1	Doxygen_MPDS _D A _R T _S DB Class List	3
3	Doxygen_MPDS_DA_RT_SDB File Index	5
3.1	Doxygen_MPDS _D A _R T _S DB File List	5
4	Doxygen_MPDS_DA_RT_SDB Class Documentation	7
4.1	dbConfig_st Struct Reference	7
4.2	dbNodeArray Class Reference	9
4.3	dbNodes Class Reference	11
4.4	StDbBroker Class Reference	15
4.5	StDbBroker::oldDescriptor Struct Reference	38
4.6	StDbWrappedMessenger Class Reference	39
5	Doxygen_MPDS_DA_RT_SDB File Documentation	41
5.1	dbConfig.h File Reference	41
5.2	DbEndian.h File Reference	42
5.3	DbFill.cxx File Reference	43
5.4	DbInit.cxx File Reference	49
5.5	dbNodeArray.h File Reference	50
5.6	dbNodes.cc File Reference	51
5.7	dbNodes.h File Reference	52
5.8	DbRead.cxx File Reference	53
5.9	DbUse.cxx File Reference	56
5.10	StDbBroker.cxx File Reference	67
5.11	StDbBroker.h File Reference	69
5.12	StDbBrokerLinkDef.h File Reference	73

5.13 StDbWrappedMessenger.cc File Reference	74
5.14 StDbWrappedMessenger.hh File Reference	75

Chapter 1

Doxxygen_MP_D_STAR_DB Hierarchical Index

1.1 Doxygen_MP_D_STAR_DB Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

dbConfig_st	7
dbNodeArray	9
dbNodes	11
StDbBroker	15
StDbBroker::oldDescriptor	38
StDbWrappedMessenger	39

Chapter 2

Doxxygen_MPDU_STARD_DB Class Index

2.1 Doxygen_MPDU_STARD_DB Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

dbConfig_st	7
dbNodeArray	9
dbNodes	11
StDbBroker	15
StDbBroker::oldDescriptor	38
StDbWrappedMessenger	39

Chapter 3

Doxxygen_MP_D_STAR_DB File Index

3.1 Doxygen_MP_D_STAR_DB File List

Here is a list of all files with brief descriptions:

dbConfig.h	41
DbEndian.h	42
DbFill.cxx	43
DbInit.cxx	49
dbNodeArray.h	50
dbNodes.cc	51
dbNodes.h	52
DbRead.cxx	53
DbUse.cxx	56
StDbBroker.cxx	67
StDbBroker.h	69
StDbBrokerLinkDef.h	73
StDbWrappedMessenger.cc	74
StDbWrappedMessenger.hh	75

Chapter 4

Doxxygen_MPDUSTAR_DB Class Documentation

4.1 dbConfig_st Struct Reference

```
#include <dbConfig.h>
```

Public Attributes

- char `tabname` [64]
- int `tabID`
- char `tabtype` [64]
- char `parname` [64]
- int `parID`

4.1.1 Member Data Documentation

4.1.1.1 int dbConfig_st::parID

Definition at line 36 of file dbConfig.h.

4.1.1.2 char dbConfig_st::parname[64]

Definition at line 35 of file dbConfig.h.

4.1.1.3 int dbConfig_st::tabID

Definition at line 33 of file dbConfig.h.

4.1.1.4 char dbConfig_st::tabname[64]

Definition at line 32 of file dbConfig.h.

4.1.1.5 char dbConfig_st::tabtype[64]

Definition at line 34 of file dbConfig.h.

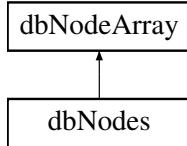
The documentation for this struct was generated from the following file:

- [dbConfig.h](#)

4.2 dbNodeArray Class Reference

```
#include <dbNodeArray.h>
```

Inheritance diagram for dbNodeArray::



Public Member Functions

- virtual ~[dbNodeArray](#) ()
- virtual int [addNode](#) (StDbNode *node, int parentID)=0
- virtual StDbNode * [getNode](#) (int index)=0
- virtual int [getParentID](#) (int index)=0
- virtual StDbNode * [getParent](#) (int index)=0
- virtual int [getNumNodes](#) ()=0
- virtual void [reset](#) ()=0
- virtual StDbNode * [next](#) ()=0

4.2.1 Constructor & Destructor Documentation

4.2.1.1 virtual dbNodeArray::~[dbNodeArray](#) () [inline, virtual]

Definition at line 33 of file dbNodeArray.h.

```
33 { };
```

4.2.2 Member Function Documentation

4.2.2.1 virtual int dbNodeArray::[addNode](#) (StDbNode * *node*, int *parentID*) [pure virtual]

Implemented in [dbNodes](#).

4.2.2.2 virtual StDbNode* dbNodeArray::[getNode](#) (int *index*) [pure virtual]

Implemented in [dbNodes](#).

4.2.2.3 virtual int dbNodeArray::[getNumNodes](#) () [pure virtual]

Implemented in [dbNodes](#).

4.2.2.4 virtual StDbNode* dbNodeArray::[getParent](#) (int *index*) [pure virtual]

Implemented in [dbNodes](#).

4.2.2.5 virtual int dbNodeArray::getParentID (int *index*) [pure virtual]

Implemented in [dbNodes](#).

4.2.2.6 virtual StDbNode* dbNodeArray::next () [pure virtual]

Implemented in [dbNodes](#).

4.2.2.7 virtual void dbNodeArray::reset () [pure virtual]

Implemented in [dbNodes](#).

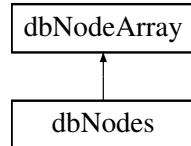
The documentation for this class was generated from the following file:

- [dbNodeArray.h](#)

4.3 dbNodes Class Reference

```
#include <dbNodes.h>
```

Inheritance diagram for dbNodes::



Public Member Functions

- [dbNodes \(\)](#)
- [~dbNodes \(\)](#)
- virtual void [deleteLists \(\)](#)
- virtual int [addNode \(StDbNode *node, int parentID\)](#)
- virtual StDbNode * [getNode \(int index\)](#)
- virtual int [getParentID \(int index\)](#)
- virtual StDbNode * [getParent \(int index\)](#)
- virtual int [getNumNodes \(\)](#)
- virtual void [reset \(\)](#)
- virtual StDbNode * [next \(\)](#)

Protected Member Functions

- void [extendParentList \(\)](#)

Protected Attributes

- [nodeVec mnodes](#)
- int [numNodes](#)
- int [curNode](#)
- int * [mpids](#)
- int [maxList](#)

4.3.1 Constructor & Destructor Documentation

4.3.1.1 dbNodes::dbNodes ()

Definition at line 31 of file dbNodes.cc.

```

31             : numNodes(0), curNode(0), maxList(500) {
32
33     mpids = new int[maxList];
34
35 }
```

4.3.1.2 dbNodes::~dbNodes () [inline]

Definition at line 67 of file dbNodes.h.

```
67 { deleteLists(); }
```

4.3.2 Member Function Documentation

4.3.2.1 int dbNodes::addNode (StDbNode * *node*, int *parentID*) [virtual]

Implements [dbNodeArray](#).

Definition at line 59 of file dbNodes.cc.

```
59
60
61 mnodes.push_back(node);
62 if(numNodes==maxList) extendParentList();
63 mpids[numNodes]=parentID;
64 numNodes++;
65
66 return numNodes-1;
67 }
```

4.3.2.2 void dbNodes::deleteLists () [virtual]

Definition at line 39 of file dbNodes.cc.

```
39
40
41 if(mpids){
42     delete [] mpids;
43     mpids = 0;
44 }
45
46 nodeVec::iterator itr;
47
48 do {
49     for(itr = mnodes.begin(); itr!=mnodes.end(); ++itr){
50         mnodes.erase(itr);
51         break;
52     }
53 } while (mnodes.begin() != mnodes.end());
54
55 }
```

4.3.2.3 void dbNodes::extendParentList () [inline, protected]

Definition at line 90 of file dbNodes.h.

```
90
91     int newMax = 2*maxList;
92     int* tmpList = new int[newMax];
93     memcpy(tmpList,mpids,maxList*sizeof(int));
94     delete [] mpids;
95     mpids = tmpList;
96     maxList = newMax;
97 }
```

4.3.2.4 StDbNode * dbNodes::getNode (int *index*) [virtual]

Implements [dbNodeArray](#).

Definition at line 72 of file dbNodes.cc.

```
72          {  
73  
74     StDbNode* node=0;  
75     if(index>-1 && index<numNodes) return mnodes[index];  
76     return node;  
77  
78 }
```

4.3.2.5 int dbNodes::getNumNodes () [inline, virtual]

Implements [dbNodeArray](#).

Definition at line 84 of file dbNodes.h.

```
84 { return numNodes; }
```

4.3.2.6 StDbNode * dbNodes::getParent (int *index*) [virtual]

Implements [dbNodeArray](#).

Definition at line 93 of file dbNodes.cc.

```
93          {  
94  
95     StDbNode* node=0;  
96     if(index>-1 && index<numNodes) return getNode(mpids[index]);  
97     return node;  
98  
99 }
```

4.3.2.7 int dbNodes::getParentID (int *index*) [virtual]

Implements [dbNodeArray](#).

Definition at line 84 of file dbNodes.cc.

```
84          {  
85  
86     if(index>-1 && index<numNodes) return mpids[index];  
87     return 0;  
88 }
```

4.3.2.8 StDbNode * dbNodes::next () [virtual]

Implements [dbNodeArray](#).

Definition at line 104 of file dbNodes.cc.

```
104          {
105
106  StDbNode* node=getNode(curNode);
107  curNode++;
108  return node;
109
110 }
```

4.3.2.9 void dbNodes::reset () [inline, virtual]

Implements [dbNodeArray](#).

Definition at line 87 of file dbNodes.h.

```
87 { curNode=0; }
```

4.3.3 Member Data Documentation

4.3.3.1 int dbNodes::curNode [protected]

Definition at line 57 of file dbNodes.h.

4.3.3.2 int dbNodes::maxList [protected]

Definition at line 60 of file dbNodes.h.

4.3.3.3 nodeVec dbNodes::mnodes [protected]

Definition at line 55 of file dbNodes.h.

4.3.3.4 int* dbNodes::mpids [protected]

Definition at line 59 of file dbNodes.h.

4.3.3.5 int dbNodes::numNodes [protected]

Definition at line 56 of file dbNodes.h.

The documentation for this class was generated from the following files:

- [dbNodes.h](#)
- [dbNodes.cc](#)

4.4 StDbBroker Class Reference

```
#include <StDbBroker.h>
```

Public Types

- `typedef St_tableDescriptor Descriptor`
- `enum EColumnType {`
- `kNAN, kFloat, kInt, kLong,`
- `kShort, kDouble, kUInt, kULong,`
- `kUShort, kUChar, kChar }`

Public Member Functions

- `StDbBroker ()`
- `virtual ~StDbBroker ()`
- `void * Use ()`
- `void SetTableFlavor (const char *flavor, int tabID, int parID)`
- `void * Use (int tabID, int parID)`
- `bool UseRunLog (StDbTable *table)`
- `char ** GetComments (St_Table *parentTable)`
- `void Fill (void *pArray, const char **ElementComment)`
- `Int_t WriteToDb (void *pArray, int tabID)`
- `Int_t WriteToDb (void *pArray, const char *fullPath, int *idList=0)`
- `StDbTable * findTable (const char *databaseName)`
- `UInt_t GetNRows ()`
- `Int_t GetBeginDate ()`
- `Int_t GetBeginTime ()`
- `const char * GetFlavor ()`
- `Int_t GetEndDate ()`
- `Int_t GetEndTime ()`
- `UInt_t GetRequestTimeStamp ()`
- `UInt_t GetBeginTimeStamp ()`
- `UInt_t GetEndTimeStamp ()`
- `UInt_t GetProdTime ()`
- `Bool_t IsZombie ()`
- `StTableDescriptorI * GetTableDescriptor ()`
- `void loadOldDescriptor ()`
- `void SetDateTime (Int_t date, Int_t time)`
- `void SetRunNumber (Int_t runNumber)`
- `void SetDictionary (UInt_t nElements, Descriptor *D)`
- `void SetDictionary (Descriptor *D)`
- `void SetTableName (const Char_t *table_name)`
- `void SetStructName (const Char_t *struct_name)`
- `void SetVersionName (const char *version)`
- `void SetDataBaseName (const char *dbName)`
- `void SetStructSize (UInt_t size)`
- `void SetNRows (UInt_t nRows)`

- void [SetBeginDate](#) (Int_t BeginDate)
- void [SetBeginTime](#) (Int_t BeginTime)
- void [SetEndDate](#) (Int_t EndDate)
- void [SetEndTime](#) (Int_t EndTime)
- void [SetRequestTimeStamp](#) (UInt_t uptime)
- void [SetBeginTimeStamp](#) (UInt_t uptime)
- void [SetEndTimeStamp](#) (UInt_t uptime)
- void [SetProdTime](#) (UInt_t ptime)
- void [SetFlavor](#) (const char *flavor)
- void [SetZombie](#) (Bool_t zombie)
- void [setVerbose](#) (int isVerbose)
- void [printStatistics](#) ()
- void [CloseAllConnections](#) ()
- dbConfig_st * [InitConfig](#) (const char *configName, int &numRows, char *versionName=0)

Static Public Member Functions

- const Char_t * [GetTypeName](#) (EColumnType type)
- int [DbInit](#) (const char *)

Public Attributes

- StDbManager * [mgr](#)

Protected Member Functions

- dbConfig_st * [buildConfig](#) (int &numRows)
- int [buildNodes](#) (StDbConfigNode *node, int pID)
- void [makeDateTime](#) (const char *dateTime, Int_t &iDate, Int_t &iTime)

Protected Attributes

- StDbTable * [m_node](#)
- oldDescriptor * [m_descriptor](#)
- Descriptor * [mdescriptor](#)
- Char_t * [m_structName](#)
- Char_t * [m_tableName](#)
- UInt_t [m_sizeOfStruct](#)
- UInt_t [m_nElements](#)
- UInt_t [m_nRows](#)
- Int_t [m_DateTime](#) [2]
- Int_t [m_BeginDate](#)
- Int_t [m_BeginTime](#)
- Int_t [m_EndDate](#)
- Int_t [m_EndTime](#)
- UInt_t [m_beginTimeStamp](#)
- UInt_t [m_endTimeStamp](#)
- UInt_t [m_requestTimeStamp](#)

- Int_t [m_runNumber](#)
- char * [m_tableVersion](#)
- char * [m_database](#)
- char * [m_ParentType](#)
- int [m_isVerbose](#)
- dbNodeArray * [m_Nodes](#)
- StDbConfigNode * [m_Tree](#)
- char * [m_flavor](#)
- unsigned int [m_prodTime](#)
- Bool_t [m_isZombie](#)

4.4.1 Member Typedef Documentation

4.4.1.1 [typedef St_tableDescriptor StDbBroker::Descriptor](#)

Definition at line 131 of file StDbBroker.h.

4.4.2 Member Enumeration Documentation

4.4.2.1 [enum StDbBroker::EColumnType](#)

Enumeration values:

kNAN

kFloat

kInt

kLong

kShort

kDouble

kUInt

kULong

kUShort

kUChar

kChar

Definition at line 128 of file StDbBroker.h.

```
128      {kNAN, kFloat, kInt, kLong, kShort, kDouble, kUInt  
129      ,kULong, kUShort, kUChar, kChar };
```

4.4.3 Constructor & Destructor Documentation

4.4.3.1 [StDbBroker::StDbBroker \(\)](#)

Definition at line 230 of file StDbBroker.cxx.

```

230 : m_structName(0), m_tableName(0), m_requestTimeStamp(0), m_tableVersion(0), m_c
231
232 m_runNumber=0;
233 m_node = 0;
234 mgr=StDbManager::Instance();
235 StDbMessService* ms=new StDbWrappedMessenger();
236 mgr->setMessenger(ms);
237
238 }

```

4.4.3.2 StDbBroker::~StDbBroker () [virtual]

Definition at line 241 of file StDbBroker.cxx.

```

241 {
242     printStatistics();
243     if(m_tableName) delete [] m_tableName;
244     if(m_structName) delete [] m_structName;
245     if(m_tableVersion) delete [] m_tableVersion;
246     if(m_database) delete [] m_database;
247     if(m_flavor) delete [] m_flavor;
248     if(m_Nodes) delete m_Nodes;
249     if(m_Tree) delete m_Tree;
250     if(mgr) delete mgr;
251 }

```

4.4.4 Member Function Documentation

4.4.4.1 dbConfig_st * StDbBroker::buildConfig (int & numRows) [protected]

Definition at line 832 of file StDbBroker.cxx.

```

832 {
833
834 dbConfig_st* cTab= 0;
835 m_Nodes->reset();
836 int numNodes = m_Nodes->getNumNodes();
837
838 StDbNode* node;
839 StDbNode* parent;
840 char* parName;
841 char* nodeName;
842 char* id;
843 unsigned int parsizesizeof(cTab[0].parname)-1;
844 unsigned int tabsizesizeof(cTab[0].tabname)-1;
845 unsigned int typsizesizeof(cTab[0].tabtype)-1;
846 int parID;
847 int cRow;
848
849     numRows = numNodes;
850
851     cTab=(dbConfig_st*)calloc(numRows,sizeof(dbConfig_st));
852     node = m_Nodes->getNode(0);
853     strncpy(cTab[0].tabname,node->printName(),tabsizesizeof);
854     cTab[0].tabname[tabsizesizeof]='\0';
855     strncpy(cTab[0].tabtype,".node",typsizesizeof);
856     cTab[0].tabtype[typsizesizeof]='\0';
857     cTab[0].parID=cTab[0].tabID=0;
858     cRow = 1;
859
860     if(m_ParentType){

```

```

861     strncpy(cTab[0].parname,m_ParentType,parsize);
862 } else {
863     strncpy(cTab[0].parname,node->printName(),parsize);
864 }
865     cTab[0].parname[parsize]='\0';
866
867 for(int i=1; i<numNodes;i++){
868
869     node = m_Nodes->getNode(i);
870     parent= m_Nodes->getParent(i);
871     parID = m_Nodes->getParentID(i);
872
873     nodeName = node->printName();
874     parName = parent->printName();
875
876     strncpy(cTab[cRow].parname,parName,parsize);
877     cTab[cRow].parname[parsize]='\0';
878     strncpy(cTab[cRow].tabname,nodeName,tabsize);
879     cTab[cRow].tabname[tabsize]='\0';
880
881     id=cTab[cRow].tabtype; *id='.'; id++;
882     if(node->IsTable()){
883         strcpy(id,((StDbTable*)node)->printCstructName());
884     } else {
885         strcpy(id,"node");
886     }
887     cTab[cRow].tabtype[tysize]='\0';
888
889     cTab[cRow].tabID=i;
890     cTab[cRow].parID=parID;
891     cRow++;
892 }
893
894 if(m_isVerbose){
895     cout <<"***** Will print dbConfig table " << endl;
896     cout <<"***** End print dbConfig table " << endl;
897     for(int k=0; k<numRows; k++) {
898         cout << "row "<<k<<" name =" << cTab[k].tabname<< " tid= "<< cTab[k].tabID;
899         cout << " type = "<<cTab[k].tabtype;
900         cout << " parent =" << cTab[k].parname << " pid= " << cTab[k].parID<< endl;
901     }
902     cout <<"***** End print dbConfig table " << endl;
903     cout <<"***** " << endl;
904 }
905
906 return cTab;
907 }
```

4.4.4.2 int StDbBroker::buildNodes (StDbConfigNode * *node*, int *pID*) [protected]

Definition at line 793 of file StDbBroker.cxx.

```

793
794
795 if(!parentNode) return 0;
796 if(!m_Nodes) {
797     m_Nodes=new dbNodes;
798     m_Nodes->addNode(parentNode,0);
799 }
800
801 int cID;
802
803 // check for tables in this Node
804     if( (parentNode->hasData()) ){
```

```

805     StDbTableIter* itr = parentNode->getStDbTableIter();
806     while(!itr->done())cID=m_Nodes->addNode(itr->next(),pID);
807     delete itr;
808 }
809
810     StDbConfigNode* next;
811
812 // check for children of this Node
813 if((parentNode->hasChildren())){
814     next=parentNode->getFirstChildNode();
815     cID=m_Nodes->addNode(next,pID);
816     if(!buildNodes(next, cID))return 0;
817 }
818
819 // check for siblings of this Node
820 int parID;
821 if( (next=parentNode->getNextNode()) ){
822     parID=m_Nodes->getParentID(pID);
823     cID=m_Nodes->addNode(next,parID);
824     if(!buildNodes(next, cID))return 0;
825 }
826
827 return 1;
828 }
```

4.4.4.3 void StDbBroker::CloseAllConnections ()

Definition at line 260 of file StDbBroker.cxx.

```

260 {
261     if(mgr)mgr->closeAllConnections();
262 };
```

4.4.4.4 int StDbBroker::DbInit (const char *) [static]

Definition at line 910 of file StDbBroker.cxx.

```
910 {    return ::DbInit(dbName) ; }
```

4.4.4.5 void StDbBroker::Fill (void *pArray, const char **ElementComment)

Definition at line 265 of file StDbBroker.cxx.

```

266 {
267     if ( m_nElements==0 ) return;
268     //char **Comments = new char*[m_nElements];
269     //TString Comment;
270
271     UInt_t i;
272     for (i=0;i<m_nElements;i++) {
273
274         if(m_descriptor[i].fDimensions>1)
275         {
276             cerr<<"dim>1, can't handle yet" << endl;
277             return;
278         }
279 }
```

```

280     m_descriptor[i].fColumnName[31]='\0';
281 }
282
283 Int_t date, time;
284 //VP TDatetime::GetDateTime(m_DateTime, date, time);
285 date = m_DateTime[0]; time = m_DateTime[1];
286 uint datetime[4]={0,0,0,0};
287 datetime[0]=date;
288 datetime[1]=time;
289
290 ::DbFill(datetime, (const char*) m_tableName, (const char*) m_structName, m_nElements,m_descriptor,C
291
292 delete [] Comments;
293 }
```

4.4.4.6 StDbTable * StDbBroker::findTable (const char * *databaseName*)

Definition at line 644 of file StDbBroker.cxx.

```

644 {
645
646 StDbTable* table=0;
647 StDbConfigNode* node= mgr->initConfig(databaseName);
648 StDbTable* tmp=node->addDbTable(m_tableName);
649 if(tmp)table=new StDbTable(*tmp);
650 delete node;
651
652 return table;
653 }
```

4.4.4.7 Int_t StDbBroker::GetBeginDate () [inline]

Definition at line 213 of file StDbBroker.h.

```
213 {return m_BeginDate; }
```

4.4.4.8 Int_t StDbBroker::GetBeginTime () [inline]

Definition at line 214 of file StDbBroker.h.

```
214 {return m_BeginTime; }
```

4.4.4.9 UInt_t StDbBroker::GetBeginTimeStamp () [inline]

Definition at line 219 of file StDbBroker.h.

```
219 {return m_beginTimeStamp; }
```

4.4.4.10 char StDbBroker::GetComments (St_Table * *parentTable*)****4.4.4.11 Int_t StDbBroker::GetEndDate () [inline]**

Definition at line 216 of file StDbBroker.h.

```
216 {return m_EndDate; }
```

4.4.4.12 Int_t StDbBroker::GetEndTime () [inline]

Definition at line 217 of file StDbBroker.h.

```
217 {return m_EndTime; }
```

4.4.4.13 UInt_t StDbBroker::GetEndTimeStamp () [inline]

Definition at line 220 of file StDbBroker.h.

```
220 {return m_endTimeStamp; }
```

4.4.4.14 const char * StDbBroker::GetFlavor ()

Definition at line 296 of file StDbBroker.cxx.

```
297 {  
298     return (m_node)? m_node->getFlavor():0;  
299 }
```

4.4.4.15 UInt_t StDbBroker::GetNRows () [inline]

Definition at line 212 of file StDbBroker.h.

```
212 {return m_nRows; }
```

4.4.4.16 UInt_t StDbBroker::GetProdTime () [inline]

Definition at line 221 of file StDbBroker.h.

```
221 {return m_prodTime; }
```

4.4.4.17 UInt_t StDbBroker::GetRequestTimeStamp () [inline]

Definition at line 218 of file StDbBroker.h.

```
218 {return m_requestTimeStamp; }
```

4.4.4.18 StTableDescriptorI * StDbBroker::GetTableDescriptor ()

Definition at line 303 of file StDbBroker.cxx.

```
303
304
305 StDbBuffer buff;
306 StDbTableDescriptor* descriptor = new StDbTableDescriptor();
307 unsigned int numElements = mdescriptor->NumberOfColumns();
308
309 for(int i=0;i<(int)numElements;i++){
310
311     buff.WriteScalar(mdescriptor->ColumnName(i),"name");
312
313     // array designation & lengths
314     if(!mdescriptor->Dimensions(i)){
315         buff.WriteScalar("1","length");
316     } else {
317         StString os;
318         const unsigned int* index = mdescriptor->IndexArray(i);
319         for(int k=0; k<(int)mdescriptor->Dimensions(i)-1;k++)
320             os<<index[k]<< ",";
321         os<<index[mdescriptor->Dimensions(i)-1];
322         const char* lengthString = (os.str()).c_str();
323         buff.WriteScalar(lengthString,"length");
324         //     os.freeze(0);
325         //     delete [] lengthString;
326     }
327
328     // position in struct
329     buff.WriteScalar(i+1,"position");
330
331     // Type identification
332     switch ((EColumnType)mdescriptor->ColumnType(i)) {
333     case kFloat:
334     {
335         buff.WriteScalar("float","type");
336         break;
337     }
338     case kInt:
339     {
340         buff.WriteScalar("int","type");
341         break;
342     }
343     case kLong:
344     {
345         buff.WriteScalar("long","type");
346         break;
347     }
348     case kShort:
349     {
350         buff.WriteScalar("short","type");
351         break;
352     }
353     case kDouble:
354     {
355         buff.WriteScalar("double","type");
356         break;
357     }
358     case kUInt:
359     {
360         buff.WriteScalar("uint","type");
361         break;
362     }
363     case kULong:
364     {
365         buff.WriteScalar("ulong","type");
366     }
```

```

366         break;
367     }
368     case kUShort:
369     {
370         buff.WriteScalar("ushort", "type");
371         break;
372     }
373     case kUChar:
374     {
375         buff.WriteScalar("uchar", "type");
376         break;
377     }
378     case kChar:
379     {
380         buff.WriteScalar("char", "type");
381         break;
382     }
383     default:
384     {
385         break;
386     }
387 }
388
389 descriptor->fillElement(&buff, 0); // 0 means use this .... don't check
390                                         // about internal schemaID's
391 buff.Raz();
392
393 }
394
395 return descriptor;
396 }
```

4.4.4.19 const Char_t* StDbBroker::GetTypeName (**EColumnType type**) [inline, static]

Definition at line 227 of file StDbBroker.h.

```

227
228     switch (type)
229     {
230         case kFloat:   return "float";
231         case kInt:    return "int";
232         case kLong:   return "long";
233         case kShort:  return "short";
234         case kDouble: return "double";
235         case kChar:   return "char";
236             //temporarily
237         case kUInt:   return "int";
238         case kULong:  return "long";
239         case kUShort: return "short";
240         case kUChar:  return "char";
241
242         case kNAN:    return "";
243         default:     return "";
244     }
245 }
```

4.4.4.20 dbConfig_st * StDbBroker::InitConfig (const char * configName, int & numRows, char * versionName = 0)

Definition at line 733 of file StDbBroker.cxx.

```

734 {
735
736     if(m_Nodes){
737         delete m_Nodes;
738         m_Nodes = 0;
739     }
740
741     if(m_Tree) delete m_Tree;
742
743     char* dbTypeName=0;
744     char* dbDomainName=0;
745     if(m_ParentType) delete [] m_ParentType;
746     m_ParentType = 0;
747
748     if(mgr->get DataBaseInfo(configName, dbTypeName, dbDomainName)){
749         if(strcmp(dbDomainName, "Star")!=0){
750             int tlen = strlen(dbTypeName);
751             m_ParentType = new char[tlen+1];
752             strcpy(m_ParentType, dbTypeName);
753         }
754     }
755
756     delete [] dbTypeName;
757     delete [] dbDomainName;
758
759     if(m_isVerbose)mgr->setVerbose(true);
760     if(!versionName){
761         m_Tree=mgr->initConfig(configName, "reconV0", 1); // 1=don't get db-descriptors
762     }else{
763         m_Tree=mgr->initConfig(configName, versionName, 1); // 1=don't get db-descriptors
764     }
765
766     dbConfig_st* configTable = 0;
767     if(!m_Tree) return configTable;
768
769     if(m_prodTime!=0)m_Tree->setProdTime(m_prodTime);
770     if(m_flavor)m_Tree->setFlavor(m_flavor);
771
772     if(m_isVerbose){
773         cout << "*****" << endl;
774         cout << "*** Will Print the Tree " << endl;
775         bool verbCheck = mgr->IsVerbose();
776         if(!verbCheck)mgr->setVerbose(true);
777         m_Tree->printTree(0);
778         if(!verbCheck)mgr->setVerbose(false);
779         cout << "*** End Print the Tree " << endl;
780         cout << "*****" << endl;
781     };
782     numRows = 0;
783
784     if(!buildNodes(m_Tree,0)) return configTable;
785
786     //numRows = m_Nodes->getNumNodes()-1;
787     return buildConfig(numRows);
788 }
```

4.4.4.21 Bool_t StDbBroker::IsZombie () [inline]

Definition at line 222 of file StDbBroker.h.

```
222 {return m_isZombie; }
```

4.4.4.22 void StDbBroker::loadOldDescriptor () [inline]

Definition at line 226 of file StDbBroker.h.

```
226 { };
```

4.4.4.23 void StDbBroker::makeDateTime (const char * *dateTime*, Int_t & *iDate*, Int_t & *iTime*) [protected]

Definition at line 512 of file StDbBroker.cxx.

```
512 {  
513  
514     char* tmp1 = new char[strlen(dateTime)+1];  
515     char* tmp2 = new char[strlen(dateTime)+1];  
516     strcpy(tmp1,dateTime);  
517     strcpy(tmp2,tmp1); tmp1[8]='\0';tmp2+=8;  
518  
519     iDate = atoi(tmp1);  
520     iTime = atoi(tmp2);  
521     delete [] tmp1; tmp2-=8; delete [] tmp2;  
522  
523 }
```

4.4.4.24 void StDbBroker::printStatistics ()

Definition at line 254 of file StDbBroker.cxx.

```
254 {  
255     if(m_Tree)m_Tree->printNumberStats();  
256     if(mgr) mgr->printTimeStats();  
257 }
```

4.4.4.25 void StDbBroker::SetBeginDate (Int_t *BeginDate*) [inline]

Definition at line 272 of file StDbBroker.h.

```
272 {m_BeginDate = BeginDate;}
```

4.4.4.26 void StDbBroker::SetBeginTime (Int_t *BeginTime*) [inline]

Definition at line 273 of file StDbBroker.h.

```
273 {m_BeginTime = BeginTime;}
```

4.4.4.27 void StDbBroker::SetBeginTimeStamp (UInt_t *utime*) [inline]

Definition at line 277 of file StDbBroker.h.

```
277 {m_beginTimeStamp = utime; }
```

4.4.4.28 void StDbBroker::SetDataBaseName (const char * *dbName*) [inline]

Definition at line 265 of file StDbBroker.h.

```
266             {if(m_database) delete [] m_database;
267             m_database= new char[strlen(dbName)+1];
268             strcpy(m_database,dbName);}
```

4.4.4.29 void StDbBroker::SetDateTime (Int_t *date*, Int_t *time*)

Definition at line 400 of file StDbBroker.cxx.

```
401 {
402     // 20000127 002502
403     m_DateTime[0] = date;
404     m_DateTime[1]= time;
405
406     StString ds;
407     StString ts;
408
409     ts<<m_DateTime[1];
410     int len = (ts.str()).length();
411     ds<<m_DateTime[0];
412     for(int i=0;i<6-len;i++)ds<<"0";
413     ds<<m_DateTime[1];
414
415     const char* dateTime = (ds.str()).c_str();
416     mgr->setRequestTime(dateTime);
417 }
```

4.4.4.30 void StDbBroker::SetDictionary (Descriptor * *D*) [inline]

Definition at line 251 of file StDbBroker.h.

```
252             {if (D) { mdescriptor = D;m_nElements=D->NumberOfColumns();}}
```

4.4.4.31 void StDbBroker::SetDictionary (UInt_t *nElements*, Descriptor * *D*) [inline]

Definition at line 249 of file StDbBroker.h.

```
250             {m_nElements=nElements; mdescriptor = D;}
```

4.4.4.32 void StDbBroker::SetEndDate (Int_t *EndDate*) [inline]

Definition at line 274 of file StDbBroker.h.

```
274 {m_EndDate = EndDate; }
```

4.4.4.33 void StDbBroker::SetEndTime (Int_t *EndTime*) [inline]

Definition at line 275 of file StDbBroker.h.

```
275 {m_EndTime = EndTime; }
```

4.4.4.34 void StDbBroker::SetEndTimeStamp (UInt_t *utime*) [inline]

Definition at line 278 of file StDbBroker.h.

```
278 {m_endTimeStamp = utime; }
```

4.4.4.35 void StDbBroker::SetFlavor (const char **flavor*)

Definition at line 426 of file StDbBroker.cxx.

```
426 {  
427  
428     if(!flavor) return;  
429     m_flavor = new char[strlen(flavor)+1];  
430     strcpy(m_flavor,flavor);  
431     if(m_Tree)m_Tree->setFlavor(m_flavor);  
432 }  
433 }
```

4.4.4.36 void StDbBroker::SetNRows (UInt_t *nRows*) [inline]

Definition at line 271 of file StDbBroker.h.

```
271 {m_nRows = nRows; }
```

4.4.4.37 void StDbBroker::SetProdTime (UInt_t *ptime*)

Definition at line 420 of file StDbBroker.cxx.

```
420 {  
421     if(m_Tree)m_Tree->setProdTime(ptime);  
422     m_prodTime = ptime;  
423 }
```

4.4.4.38 void StDbBroker::SetRequestTimeStamp (UInt_t *utime*) [inline]

Definition at line 276 of file StDbBroker.h.

```
276 {m_requestTimeStamp = utime; }
```

4.4.4.39 void StDbBroker::SetRunNumber (Int_t *runNumber*) [inline]

Definition at line 248 of file StDbBroker.h.

```
248 {m_runNumber=runNumber;};
```

4.4.4.40 void StDbBroker::SetStructName (const Char_t * *struct_name*) [inline]

Definition at line 257 of file StDbBroker.h.

```
258 {if(m_structName) delete [] m_structName;
259 m_structName=new char[strlen(struct_name)+1];
260 strcpy(m_structName,struct_name);};
```

4.4.4.41 void StDbBroker::SetStructSize (UInt_t *size*) [inline]

Definition at line 270 of file StDbBroker.h.

```
270 {m_sizeOfStruct=size;};
```

4.4.4.42 void StDbBroker::SetTableFlavor (const char * *flavor*, int *tabID*, int *parID*)

Definition at line 436 of file StDbBroker.cxx.

```
437 {
438     StDbNode* anode = m_Nodes->getNode(tabID);
439     StDbTable* node=dynamic_cast<StDbTable*>(anode);
440
441     if(!node) return;
442     node->setFlavor(flavor);
443 }
```

4.4.4.43 void StDbBroker::SetTableName (const Char_t * *table_name*) [inline]

Definition at line 253 of file StDbBroker.h.

```
254 {if(m_tableName) delete [] m_tableName;
255 m_tableName=new char[strlen(table_name)+1];
256 strcpy(m_tableName,table_name);};
```

4.4.4.44 void StDbBroker::setVerbose (int *isVerbose*) [inline]

Definition at line 285 of file StDbBroker.h.

```
285 { m_isVerbose = isVerbose; }
```

4.4.4.45 void StDbBroker::SetVersionName (const char * *version*) [inline]

Definition at line 261 of file StDbBroker.h.

```
262             {if(m_tableVersion) delete [] m_tableVersion;
263             m_tableVersion= new char[strlen(version)+1];
264             strcpy(m_tableVersion,version);}
```

4.4.4.46 void StDbBroker::SetZombie (Bool_t *zombie*) [inline]

Definition at line 281 of file StDbBroker.h.

```
281 { m_isZombie=true; }
```

4.4.4.47 void * StDbBroker::Use (int *tabID*, int *parID*)

Definition at line 447 of file StDbBroker.cxx.

```
448 {
449
450     // This is an "Offline" requirement of only 31 char per element name
451     // UInt_t i;
452     // for (i=0;i<m_nElements;i++) {
453     //     m_descriptor[i].fColumnName[31]='\0';
454     // }
455     void* pData = 0;
456     m_nRows = 0;
457     SetNRows(0);
458     SetBeginDate(19950101);
459     SetBeginTimeStamp(788918400);
460     SetBeginTime(0);
461     SetEndDate(20380101);
462     SetEndTimeStamp(2145916799);
463     SetEndTime(0);
464     SetZombie(false);
465
466     //Store the the TTABLE padded size
467     StDbTableDescriptor* TD = new StDbTableDescriptor();
468     TD->storeRowSize(m_sizeOfStruct);
469     delete TD;
470
471     StDbNode* anode = m_Nodes->getNode(tabID);
472     m_node=dynamic_cast<StDbTable*>(anode);
473
474     if(!m_node) return pData;
475     if(!m_node->hasDescriptor())m_node->setDescriptor(GetTableDescriptor());
476
477     // I would do this in a separate function but this would require
478     // redoing it all with
479
480     bool fetchStatus;
481     if(m_node->getDbType() == dbRunLog &&
482         m_node->getDbDomain() != dbStar &&
483         m_runNumber>1000000 ){
484         fetchStatus=UseRunLog(m_node);
485     } else {
486         fetchStatus=mgr->fetchDbTable(m_node);
487     }
488 }
```

```

489 // success or failure yields an endtime ... so get it.
490
491 if(fetchStatus){
492
493     char* thisTime;
494     m_endTimeStamp = m_node->getEndTime();
495     thisTime = m_node->getEndDateTime();
496     makeDateTime(thisTime,m_EndDate,m_EndTime);
497     m_nRows= m_node->GetNRows();
498     pData = m_node->GetTableCpy(); // gives the "malloc'd version"
499
500     m_beginTimeStamp = m_node->getBeginTime();
501     thisTime = m_node->getBeginDateTime();
502     makeDateTime(thisTime,m_BeginDate,m_BeginTime);
503
504 } else {
505     SetZombie(true);
506 }
507
508 return pData;
509 }
```

4.4.4.48 void * StDbBroker::Use ()

Definition at line 656 of file StDbBroker.cxx.

```

657 {
658 //convert event datetime to date and time
659 Int_t date, time;
660 //VP TDatetime::GetDateTime(m_DateTime, date, time);
661 date = m_DateTime[0]; time = m_DateTime[1];
662 uint datetime[4]={0,0,0,0};
663 datetime[0]=date;
664 datetime[1]=time;
665 uint nRows=0;
666
667
668 UInt_t i;
669 for (i=0;i<m_nElements;i++) {
670     m_descriptor[i].fColumnName[31]='\0';
671 }
672
673 // Check if request is a "hierarchy" : if so send request to "params" DB
674 char* id = strstr(m_tableName,"_hierarchy");
675 if(id){
676     char* tmpName = new char[strlen(m_tableName)+1];
677     strcpy(tmpName,m_tableName);
678     char* id2 = strstr(tmpName,"_hierarchy");
679     *id2 = '\0';
680     if(strcmp(tmpName,"Calib")==0){
681         m_database = new char[strlen("Calibrations_tpc")+1];
682         strcpy(m_database,"Calibrations_tpc");
683     } else if(strstr(tmpName,"Geom")){
684         m_database = new char[strlen("Geometry_tpc")+1];
685         strcpy(m_database,"Geometry_tpc");
686     } else if(strstr(tmpName,"RunParam")){
687         m_database = new char[strlen("RunParams_tpc")+1];
688         strcpy(m_database,"RunParams_tpc");
689     } else {
690         m_database = new char[strlen("params")+1];
691         strcpy(m_database,"params");
692     }
693     *id2='_';
694     delete [] tmpName;
```

```

695     }
696
697     void *pDbData;
698
699     if(id || strcmp(m_database,"params")==0){
700
701         cout << "Looking for Table "<<m_tableName<<" In Db= "<<m_database<<endl;
702
703     pDbData = ::DbUse(&nRows, datetime, (const char*)m_tableName,(const char*)m_structName,m_nElements,m_nElements);
704
705     } else {
706
707     pDbData = ::DbRead(&nRows, datetime,(const char*)m_tableName,(const char*)m_structName,m_nElements,m_nElements);
708
709    }
710
711    if (pDbData==NULL)
712    {
713        SetNRows(0);
714        SetBeginDate(19950101);
715        SetBeginTime(0);
716        SetEndDate(20380101);
717        SetEndTime(0);
718    }
719    else
720    {
721        SetNRows((UInt_t)nRows);
722        SetBeginDate(datetime[0]);
723        SetBeginTime(datetime[1]);
724        SetEndDate (datetime[2]);
725        SetEndTime (datetime[3]);
726    }
727
728    return pDbData;
729 }
```

4.4.4.49 bool StDbBroker::UseRunLog (StDbTable * *table*)

Definition at line 526 of file StDbBroker.cxx.

```

526
527
528     unsigned int prodTime=table->getProdTime();
529     StString rq;
530     rq<< " where runNumber="<<m_runNumber;
531
532     if(prodTime==0){
533         rq<< " AND deactivate=0 ";
534     } else {
535         rq<< " AND (deactive=0 OR deactivate>="<<prodTime<< ")";
536         rq<< " AND unix_timestamp(entryTime)<="<<prodTime;
537     }
538
539     bool fetchStatus=mgr->fetchDbTable(table,(char*)(rq.str()).c_str());
540
541
542 return fetchStatus;
543 }
```

4.4.4.50 Int_t StDbBroker::WriteToDb (void * *pArray*, const char * *fullPath*, int * *idList* = 0)

Definition at line 577 of file StDbBroker.cxx.

```

577
578 #define __METHOD__ "WriteToDb(pArray,fullPath,idList)"
579     StString em;
580
581     if(!pArray || !fullPath) {
582         em<<" Write Failed:: either data-array or path is incomplete";
583         return mgr->printInfo((em.str()).c_str(),dbMErr,__LINE__,__CLASS__,__METHOD__);
584     }
585
586     char* path=new char[strlen(fullPath)+1];
587     strcpy(path,fullPath);
588
589     // chop off a trailing "/" if found
590     char* tmp=path;
591     tmp+=strlen(path)-1;
592     if(*tmp=='/')*tmp='\0';
593
594     // Algorithm is if path = "A/B/C/D", we try to connect to
595     // databases "D", "C_D" (& "C") , "B_C" ( & "B"), "A_B" (& "A")
596     // quiting when table is found. Note, by design if database
597     // "C_D" does not exist, the DB-API tries to find database "C".
598
599     char* a1=path;
600     char* a2;
601     char** aword = new char*[20];
602
603     int icount=0;
604     aword[icount]=a1;
605     while((a2=strstr(a1,"/"))){
606         *a2='\0'; a2++;
607         icount++;
608         aword[icount]=a2;
609         a1=a2;
610     }
611
612     char tmpName[128];
613     char* dbName =0;
614     StDbTable* table = 0;
615     for(int i=icount;i>0;i--){
616         if(i==icount){
617             dbName=aword[i];
618         } else {
619             sprintf(tmpName,"%s_%s",aword[i-1],aword[i]);
620             dbName=(char*)tmpName;
621         }
622         table=findTable(dbName);
623         if(table)break;
624     }
625
626     if(!table){
627         em<<"Write Failed table=<<m_tableName<<" not found in db="<<dbName;
628         delete [] path;
629         return mgr->printInfo((em.str()).c_str(),dbMErr,__LINE__,__CLASS__,__METHOD__);
630     }
631
632     table->setDescriptor(GetTableDescriptor());
633     table->SetTable((char*)pArray,m_nRows,idList);
634     mgr->setStoreTime(mgr->getUnixCheckTime());
635     bool iswritten=mgr->storeDbTable(table);
636     delete table;
637
638     return (iswritten) ? 1 : 0;
639 #undef __METHOD__
640 }
```

4.4.4.51 Int_t StDbBroker::WriteToDb (void * *pArray*, int *tabID*)

Definition at line 546 of file StDbBroker.cxx.

```

546
547 #define __METHOD__ "WriteToDb(pArray,tabID)"
548
549     StString em;
550     if(!pArray || tabID==0) {
551         em<<" Write Failed -> either data-array or tableID is incomplete";
552         return mgr->printInfo((em.str()).c_str(),dbMErr,__LINE__,__CLASS__,__METHOD__);
553     }
554     if(!m_Nodes){
555         em<<"Write Failed -> incomplete table context. Try InitConfig() 1st";
556         return mgr->printInfo((em.str()).c_str(),dbMErr,__LINE__,__CLASS__,__METHOD__);
557     }
558     StDbNode* anode= m_Nodes->getNode(tabID);
559     StDbTable* table=dynamic_cast<StDbTable*>(anode);
560     if(!table){
561         em<<"Write Failed -> tableID=<<tabID<<" is not known ";
562         return mgr->printInfo((em.str()).c_str(),dbMErr,__LINE__,__CLASS__,__METHOD__);
563     }
564     if(!table->hasDescriptor())table->setDescriptor(GetTableDescriptor());
565     table->SetTable((char*)pArray,m_nRows);
566
567     // WARNING :: A Cludge -> StDbManager has separate
568     // 'store' & 'request' times whilr StDbBroker does not
569     mgr->setStoreTime(mgr->getUnixCheckTime());
570     if(!mgr->storeDbTable(table))return 0;
571     return 1;
572
573 #undef __METHOD__
574 }
```

4.4.5 Member Data Documentation

4.4.5.1 Int_t StDbBroker::m_BeginDate [protected]

Definition at line 156 of file StDbBroker.h.

4.4.5.2 Int_t StDbBroker::m_BeginTime [protected]

Definition at line 157 of file StDbBroker.h.

4.4.5.3 UInt_t StDbBroker::m_beginTimeStamp [protected]

Definition at line 161 of file StDbBroker.h.

4.4.5.4 char* StDbBroker::m_database [protected]

Definition at line 168 of file StDbBroker.h.

4.4.5.5 Int_t StDbBroker::m_DateTime[2] [protected]

Definition at line 154 of file StDbBroker.h.

4.4.5.6 `oldDescriptor* StDbBroker::m_descriptor` [protected]

Definition at line 146 of file StDbBroker.h.

4.4.5.7 `Int_t StDbBroker::m_EndDate` [protected]

Definition at line 158 of file StDbBroker.h.

4.4.5.8 `Int_t StDbBroker::m_EndTime` [protected]

Definition at line 159 of file StDbBroker.h.

4.4.5.9 `UInt_t StDbBroker::m_endTimeStamp` [protected]

Definition at line 162 of file StDbBroker.h.

4.4.5.10 `char* StDbBroker::m_flavor` [protected]

Definition at line 175 of file StDbBroker.h.

4.4.5.11 `int StDbBroker::m_isVerbose` [protected]

Definition at line 171 of file StDbBroker.h.

4.4.5.12 `Bool_t StDbBroker::m_isZombie` [protected]

Definition at line 180 of file StDbBroker.h.

4.4.5.13 `UInt_t StDbBroker::m_nElements` [protected]

Definition at line 151 of file StDbBroker.h.

4.4.5.14 `StDbTable* StDbBroker::m_node` [protected]

Definition at line 145 of file StDbBroker.h.

4.4.5.15 `dbNodeArray* StDbBroker::m_Nodes` [protected]

Definition at line 172 of file StDbBroker.h.

4.4.5.16 `UInt_t StDbBroker::m_nRows` [protected]

Definition at line 152 of file StDbBroker.h.

4.4.5.17 char* StDbBroker::m_ParentType [protected]

Definition at line 169 of file StDbBroker.h.

4.4.5.18 unsigned int StDbBroker::m_prodTime [protected]

Definition at line 176 of file StDbBroker.h.

4.4.5.19 UInt_t StDbBroker::m_requestTimeStamp [protected]

Definition at line 163 of file StDbBroker.h.

4.4.5.20 Int_t StDbBroker::m_runNumber [protected]

Definition at line 165 of file StDbBroker.h.

4.4.5.21 UInt_t StDbBroker::m_sizeOfStruct [protected]

Definition at line 150 of file StDbBroker.h.

4.4.5.22 Char_t* StDbBroker::m_structName [protected]

Definition at line 148 of file StDbBroker.h.

4.4.5.23 Char_t* StDbBroker::m_tableName [protected]

Definition at line 149 of file StDbBroker.h.

4.4.5.24 char* StDbBroker::m_tableVersion [protected]

Definition at line 167 of file StDbBroker.h.

4.4.5.25 StDbConfigNode* StDbBroker::m_Tree [protected]

Definition at line 173 of file StDbBroker.h.

4.4.5.26 Descriptor* StDbBroker::mdescriptor [protected]

Definition at line 147 of file StDbBroker.h.

4.4.5.27 StDbManager* StDbBroker::mgr

Definition at line 291 of file StDbBroker.h.

The documentation for this class was generated from the following files:

- [StDbBroker.h](#)

- [StDbBroker.cxx](#)

4.5 StDbBroker::oldDescriptor Struct Reference

```
#include <StDbBroker.h>
```

Public Attributes

- char [fColumnName](#) [32]
- unsigned int [fIndexArray](#) [3]
- unsigned int [fOffset](#)
- unsigned int [fSize](#)
- unsigned int [fTypeSize](#)
- unsigned int [fDimensions](#)
- Int_t [fType](#)

4.5.1 Member Data Documentation

4.5.1.1 char [StDbBroker::oldDescriptor::fColumnName\[32\]](#)

Definition at line 135 of file StDbBroker.h.

4.5.1.2 unsigned int [StDbBroker::oldDescriptor::fDimensions](#)

Definition at line 140 of file StDbBroker.h.

4.5.1.3 unsigned int [StDbBroker::oldDescriptor::fIndexArray\[3\]](#)

Definition at line 136 of file StDbBroker.h.

4.5.1.4 unsigned int [StDbBroker::oldDescriptor::fOffset](#)

Definition at line 137 of file StDbBroker.h.

4.5.1.5 unsigned int [StDbBroker::oldDescriptor::fSize](#)

Definition at line 138 of file StDbBroker.h.

4.5.1.6 Int_t [StDbBroker::oldDescriptor::fType](#)

Definition at line 141 of file StDbBroker.h.

4.5.1.7 unsigned int [StDbBroker::oldDescriptor::fTypeSize](#)

Definition at line 139 of file StDbBroker.h.

The documentation for this struct was generated from the following file:

- [StDbBroker.h](#)

4.6 StDbWrappedMessenger Class Reference

```
#include <StDbWrappedMessenger.hh>
```

Public Member Functions

- [StDbWrappedMessenger \(\)](#)
- [virtual ~StDbWrappedMessenger \(\)](#)
- [virtual void printMessage \(const char *message, StDbMessLevel dbLevel, int lineNumber, const char *className, const char *methodName\)](#)
- [virtual void printMessage \(const char *message, const char *levelString, int lineNumber, const char *className, const char *methodName\)](#)

Protected Attributes

- [StMessMgr * mMessenger](#)

4.6.1 Constructor & Destructor Documentation

4.6.1.1 StDbWrappedMessenger::StDbWrappedMessenger ()

Definition at line 44 of file StDbWrappedMessenger.cc.

```
44
45     mMessenger=StMessageManager::Instance();
46 }
```

4.6.1.2 virtual StDbWrappedMessenger::~StDbWrappedMessenger () [inline, virtual]

Definition at line 34 of file StDbWrappedMessenger.hh.

```
34 {};
```

4.6.2 Member Function Documentation

4.6.2.1 void StDbWrappedMessenger::printMessage (const char * *message*, const char * *levelString*, int *lineNumber*, const char * *className*, const char * *methodName*) [virtual]

Definition at line 90 of file StDbWrappedMessenger.cc.

```
90
91
92 // 
93 // limit here of 1024 only hits us for the StDbManager in verbose mode
94 // which isn't available directly in StRoot
95 //
96
97 // char str[1024];
98 int n = strlen(message)+1000;
```

```

99     char * str = new char[n];
100
101    sprintf(str,"%s::%s line=%d %s",className,methodName,lineNumber,message);
102    mMessenger->Message(str,levelString);
103    delete [] str;
104 }

```

4.6.2.2 void StDbWrappedMessenger::printMessage (const char * *message*, StDbMessLevel *dbLevel*, int *lineNumber*, const char * *className*, const char * *methodName*) [virtual]

Definition at line 51 of file StDbWrappedMessenger.cc.

```

51
52
53 if(dbLevel<mdbLevel) return;
54
55 char lString[64];
56 switch(dbLevel){
57 case dbMDebug:
58 {
59     strcpy(lString,"I");
60     break;
61 }
62 case dbMWarn:
63 {
64     strcpy(lString,"W");
65     break;
66 }
67 case dbMConnect:
68 {
69     strcpy(lString,"I");
70     break;
71 }
72 case dbMErr:
73 {
74     strcpy(lString,"E");
75     break;
76 }
77 default:
78 {
79     strcpy(lString," ");
80     break;
81 }
82 }
83
84 printMessage(message,(const char*)lString,lineNumber,className,methodName);
85 }

```

4.6.3 Member Data Documentation

4.6.3.1 StMessMgr* StDbWrappedMessenger::mMessenger [protected]

Definition at line 29 of file StDbWrappedMessenger.hh.

The documentation for this class was generated from the following files:

- [StDbWrappedMessenger.hh](#)
- [StDbWrappedMessenger.cc](#)

Chapter 5

Doxxygen_MP_D_STAR_DB File Documentation

5.1 dbConfig.h File Reference

Classes

- struct [dbConfig_st](#)

5.2 DbEndian.h File Reference

5.3 DbFill.cxx File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <Stsstream.h>
#include <Stiostream.h>
#include "mysql.h"
#include "mysql_com.h"
#include "StDbBroker.h"
```

Functions

- char * [strmov](#) (char *dst, const char *src)
- void [DbFill](#) (uint *datetime, const char *tableName, const char *structName, uint nVar, [StDbBroker::oldDescriptor](#) *d, const char **Comments, uint nRows, uint sizeOfStruct, void *pData)

5.3.1 Function Documentation

5.3.1.1 void [DbFill](#) (uint * *datetime*, const char * *tableName*, const char * *structName*, uint *nVar*, [StDbBroker::oldDescriptor](#) * *d*, const char ** *Comments*, uint *nRows*, uint *sizeOfStruct*, void * *pData*)

Definition at line 54 of file DbFill.cxx.

```
63 {
64     //cout << "DbFill: structure: " << structName << endl;
65 //cout << "Sorry, this software release does not let users to fill the database" << endl;
66 //    return;
67
68 //    for (uint i=0;i<nVar;i++){
69 //        cout <<i<<endl;
70 //        cout <<d[i].name<<endl;
71 //        cout <<d[i].firstDimension<<endl;
72 //        cout <<d[i].offset<<endl;
73 //        cout <<d[i].dimensions<<endl;
74 //        cout <<StDbBroker::GetTypeName(d[i].type)<<endl;
75 //    }
76
77 MySQL mysql;
78 MySQL_RES *result;
79 MySQL_ROW row;
80
81 unsigned int num_fields;
82 unsigned int num_rows;
83
84 const int MAXBUF=2048;
85 char query[MAXBUF];
86 //VP char buf[MAXBUF];
87 //VP ostrstream Query(buf,MAXBUF);
88 ostrstream Query;
89 char *end;
90 char *binQuery;
91
```

```

92 //sun:
93 uint count;
94 /uint num_elem;
95 uint i, j;
96
97 //cout<<"filling: "<<tableName<<, date "<<datetime[0]<<" and time "<<datetime[1]<<endl;
98
99 //char validFrom[15];
100 char validFrom[20];
101 char time[7];
102 //TIMESTAMP format: "YYYYMMDDHHMMSS";
103 sprintf(validFrom,"% .8d",datetime[0]);
104 sprintf(time,"% .6d",datetime[1]);
105 strcat(validFrom,time);
106
107 //convert to DATETIME format: 0000-00-00 00:00:00
108
109 validFrom[19]='\0';
110 validFrom[18]=validFrom[13];
111 validFrom[17]=validFrom[12];
112 validFrom[16]=':';
113 validFrom[15]=validFrom[11];
114 validFrom[14]=validFrom[10];
115 validFrom[13]=':';
116 validFrom[12]=validFrom[9];
117 validFrom[11]=validFrom[8];
118 validFrom[10]=' ';
119 validFrom[9]=validFrom[7];
120 validFrom[8]=validFrom[6];
121 validFrom[7]='-';
122 validFrom[6]=validFrom[5];
123 validFrom[5]=validFrom[4];
124 validFrom[4]='-';
125
126 cout<<"validFrom: " <<validFrom<<endl;
127
128 // Initialize a connection handler
129
130 mysql_init(&mysql);
131
132 // Establish a connection to the MySQL database engine
133 // params - for reading, test_blob -for development
134
135 //if (!mysql_real_connect(&mysql,"duvall.star.bnl.gov","root","","params",0,NULL,0))
136 //if (!mysql_real_connect(&mysql,"duvall.star.bnl.gov","writer","","params",0,NULL,0))
137 if (!mysql_real_connect(&mysql,"localhost","stardb","","params",0,NULL,0))
138 //if (!mysql_real_connect(&mysql,"db1.star.bnl.gov","writer","","params",0,NULL,0))
139 {
140     cerr << "Failed to connect to database: Error: "
141         << mysql_error(&mysql) << endl;
142 }
143
144 //check structure name/size first
145
146 int maxVersion=0;
147 uint *existingStrID=NULL;
148 uint structureID=0;
149 uint num_struct=0;
150
151 Query.seekp(0);
152 Query << "SELECT ID,version FROM structures WHERE name=\"<<structName
153     <<\" AND sizeOfStruct=<<sizeOfStruct
154     <<\" AND nElements=<<nVar<<\" ORDER BY version"<<ends;
155
156 //cout << "database query: " << Query.str() << endl;
157
158 if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))

```

```

159  {
160      cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
161      mysql_close(&mysql);
162      return;
163  }
164 else // query succeeded, get result
165  {
166     result = mysql_store_result(&mysql);
167     if (result)
168     {
169         num_fields = mysql_num_fields(result);
170         if (num_fields!=2) cerr <<"ERROR: wrong size of struct query"<<endl;
171
172         num_struct = mysql_num_rows(result);
173         if (num_struct==0)// this is the new struct name/size to insert
174         {
175             //cout<< "struct "<<structName<<" do not exist in database" <<endl;
176             //cout << "database query: " << Query.str() << endl;
177             maxVersion=0;
178         }
179     else // this structure name/size already exists
180     {
181 //         if (num_struct>0) {
182 //             cout << "INFO: there is structure "
183 //                 << structName << endl;
184 //             //cout << "database query: " << Query.str() << endl;
185 //         }
186         existingStrID = new uint[num_struct];
187         for (uint id=0;id<num_struct;id++)
188         {
189             row = mysql_fetch_row(result);
190             existingStrID[id]=(uint)atoi(row[0]);
191             maxVersion=atoi(row[1]);
192         }
193     }
194     mysql_free_result(result);
195 }
196 else // query OK but, no result?!
197 {
198     cerr << "no result: Error: " << mysql_error(&mysql) << endl;
199     mysql_close(&mysql);
200     return;
201 }
202 }
203
204 int same=0;
205
206 for (i=0;i<num_struct;i++)
207 {
208     //same variables counter
209     count=0;
210
211     //cout << "i, num_struct, existingStrID[i]: " << i<<" "<<num_struct<<" "<<existingStrID[i] << endl;
212
213     Query.seekp(0);
214     Query << "SELECT headers.name, headers.offset, headers.nDims, headers.firstDim, headers.comment ";
215
216     //cout << "database query: " << Query.str() << endl;
217     if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
218     {
219         cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
220         mysql_close(&mysql);
221         return;
222     }
223     else // query succeeded, get result
224     {
225         result = mysql_store_result(&mysql);

```

```

226     if (result)// query OK
227     {
228         num_fields = mysql_num_fields(result);
229         if (num_fields!=5) cerr << "ERROR: wrong size of headers query" << endl;
230
231         num_rows = mysql_num_rows(result);
232
233         if (num_rows!=nVar) //size is different from this ID
234         {
235             cout<<"database structure "<<i<<" is of "<< num_rows
236                 <<" variables, this structure is "<< nVar << endl;
237             continue;
238         }
239         else //same size size: check names
240         {
241             for (j=0;j<num_rows;j++)
242             {
243                 row = mysql_fetch_row(result);
244
245                 if (strcmp(d[j].fColumnName,row[0]))
246                     break;
247
248                 //same name: check offset
249                 //cout<<"offset: \"<<d[j].offset<<"\" \\"<<row[1]<<"\"<<endl;
250                 if ( d[j].fOffset!=(unsigned int)atoi(row[1]) )
251                     break;
252
253                 //same offset: check nDims
254                 //cout<<"nDims: \"<<d[j].dimensions<<"\" \\"<<row[2]<<"\"<<endl;
255                 if ( d[j].fDimensions!=(unsigned int)atoi(row[2]) )
256                     break;
257
258                 //same nDims: check firstDim
259                 //cout<<"nDims: \"<<d[j].firstDimension<<"\" \\"<<row[3]<<"\"<<endl;
260                 if ( d[j].fIndexArray[0]!=(unsigned int)atoi(row[3]) )
261                     break;
262
263                 //same value: check comment
264
265                 // strip trailing blanks from Comments
266 //                 int jc=strlen(Comments[j]);
267 //                 while (jc != 0 && Comments[j][jc-1] == ' ') jc--;
268 //                 Comments[j][jc]='\0';
269
270                 //cout<<"Comment: \n\"<<Comments[j]<<"\"\\n\"\"<<row[4]<<"\"<<endl;
271                 //DATABASE Comments are truncated to CHAR[64]
272                 if (strncmp(Comments[j],row[4],64))//different comment, new structure?
273                 {
274                     break;
275                 }
276                 //cout<<"variable "<< j+1 <<" is the same" << endl;
277                 count++;
278             }
279
280             if (count==nVar)
281             {
282                 structureID=existingStrID[i];
283                 cout<<"structure " << structName
284                     <<" is found in database with ID "<<structureID<< endl;
285                 same=1;
286                 break;
287             }
288             //end if equal size structures
289             mysql_free_result(result);
290         }
291     else // something wrong
292     {

```

```

293         cerr << "no result, Error: " << mysql_error(&mysql) << endl;
294         mysql_close(&mysql);
295         return;
296     }
297 }
298
299 //end of loop over all existing structure IDs
300
301 if (same==0)//we have to insert this structure
302 {
303     // add struct to structures table
304
305     Query.seekp(0);
306     Query << "INSERT INTO structures SET name=\"<<structName
307             <<"\", version=<<maxVersion+1
308             <<", sizeOfStruct=<<sizeOfStruct
309             <<", nElements=<<nVar<<ends;
310
311     //cout << "database query: " << Query.str() << endl;
312     if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
313     {
314         cerr<<"ERROR: Failed to insert, " << mysql_error(&mysql) << endl;
315         cerr << "database query: " << Query.str() << endl;
316         mysql_close(&mysql);
317         return;
318     }
319     else
320     {
321         structureID = mysql_insert_id(&mysql);
322         cout << "new strID: " << structureID << endl;
323     }
324     //insert parameters here
325     for (j=0;j<nVar;j++)
326     {
327         Query.seekp(0);
328         Query << "INSERT INTO headers SET strID=\"<<structureID
329                 <<"\", name=\"<<d[j].fColumnName
330                 <<"\", type=\"<<StDbBroker::GetTypeName((StDbBroker::EColumnType)d[j].fType)
331                 <<"\", nDims=\"<<d[j].fDimensions//ENUM has to be as string
332                 <<"\", firstDim=\"<<d[j].fIndexArray[0]
333                 <<", offset=\"<<d[j].fOffset
334                 <<", comment=\"<<ends;
335
336         //we have to escape comments
337         int lenComment=strlen(Comments[j]);
338         end = strmov(query,Query.str());
339         end += mysql_escape_string(end,Comments[j],lenComment);
340         *end++ = '\0';
341
342         //if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
343         if (mysql_real_query(&mysql,query,(end - query)))
344         {
345             cerr<<"ERROR: Failed to insert, " << mysql_error(&mysql) << endl;
346             *end++ = '\0';
347             cerr << "database query: " << query << endl;
348             mysql_close(&mysql);
349             return;
350         }
351     else
352     {
353         //cout << "new parameter ID: " << mysql_insert_id(&mysql)<< endl;
354     }
355
356     }//end loop over parameters
357 } //end of same==0 if
358
359 //we have the structureID, now insert the sttable instances

```

```

360
361 uint tableID=0;
362 Query.seekp(0);
363 Query << "INSERT INTO instances SET name=\"<<tableName
364     <<"\", validFrom=\"<<validFrom
365     <<"\", strID=<<structureID
366     <<\", nrows=<<nRows<<ends;
367
368 //cout << "database query: " << Query.str() << endl;
369
370 if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
371 {
372     cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
373     cerr << "database query: " << Query.str() << endl;
374     mysql_close(&mysql);
375     return;
376 }
377 else
378 {
379     tableID = mysql_insert_id(&mysql);
380 //     cout << "new table ID: " << tableID << endl;
381 }
382
383 Query.seekp(0);
384 Query << "INSERT INTO bytes SET instanceID=<<tableID
385     <<\", bytes=\"<<ends;
386 //cout << "database query: " << Query.str() << endl;
387
388 //make space for escaped bin data of length*2+1
389
390 uint byteSize = sizeOfStruct*nRows;
391 binQuery = new char[2*byteSize+Query.pcount()];
392 end = strmov(binQuery,Query.str());
393 end += mysql_escape_string(end,(const char*)pData,byteSize);
394 *end++ = '\0';
395
396 if (mysql_real_query(&mysql,binQuery,(int) (end - binQuery)))
397 {
398     cerr << "Failed to insert binQuery: Error: "<<mysql_error(&mysql)<<endl;
399     cerr << "start of query: " << Query.str() << endl;
400     mysql_close(&mysql);
401     return;
402 }
403
404 if (existingStrID!=NULL) delete [] existingStrID;
405 delete [] binQuery;
406 // We are done with the connection, call mysql_close() to terminate it
407
408 mysql_close(&mysql);
409
410 return;
411
412 }

```

5.3.1.2 **char* strmov (char * dst, const char * src)**

5.4 DbInit.cxx File Reference

```
#include <Stiostream.h>
#include "mysql.h"
```

Functions

- int **DbInit** (const char *dbName)

5.4.1 Function Documentation

5.4.1.1 int DbInit (const char * *dbName*)

Definition at line 26 of file DbInit.cxx.

```
27 {
28
29 MySQL mysql;
30
31 mysql_init(&mysql);
32
33 //set timeout in seconds for bnl.local domain
34
35 #ifndef __sun
36 //on sun:
37 //Program received signal SIGBUS, Bus error.
38 //0xed737d68 in mysql_options ()
39
40 mysql_options(&mysql,MYSQL_OPT_CONNECT_TIMEOUT,"2");
41
42 #endif
43 // Try to establish a connection to the MySQL database engine
44
45 const char *database=dbName;
46 //only db1 is visible from rcas0202 machine
47 const char *dbHost="db1.star.bnl.gov";
48 //char *dbHost="duvall.star.bnl.gov";
49 //mysql_real_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd, const char *
50
51 if (!mysql_real_connect(&mysql,dbHost,"","",database,0,NULL,0))
52 {
53     cerr << "Failed to connect to database: "<<database<< endl;
54     cerr << "MySQL ERROR: " << mysql_error(&mysql) << endl;
55     return 1;
56 }
57
58 // We are done with the connection, call mysql_close() to terminate it
59 mysql_close(&mysql);
60
61 return 0;
62
63 }
```

5.5 dbNodeArray.h File Reference

Classes

- class [dbNodeArray](#)

5.6 dbNodes.cc File Reference

```
#include "dbNodes.h"
```

5.7 dbNodes.h File Reference

```
#include "dbNodeArray.h"
#include "StDbLib/StDbNode.hh"
#include <vector>
```

Classes

- class [dbNodes](#)

TypeDefs

- typedef vector< StDbNode * > [nodeVec](#)

5.7.1 Typedef Documentation

5.7.1.1 [typedef vector<StDbNode*> nodeVec](#)

Definition at line 47 of file dbNodes.h.

5.8 DbRead.cxx File Reference

```
#include <stdlib.h>
#include "StDbLib/StDbManager.hh"
#include "StDbBroker.h"
#include "StDbLib/StDbDefs.hh"
#include "StDbLib/StDbConfigNode.hh"
#include "StDbLib/StDbTable.h"
#include "StDbLib/StDbTableDescriptor.h"
```

Functions

- void * **DbRead** (unsigned int *nRows, unsigned int *datetime, const char *tableName, const char *structName, unsigned int nVar, unsigned int sizeOfStruct, **StDbBroker::oldDescriptor** *d, const char *database, const char *tableVersion)

5.8.1 Function Documentation

5.8.1.1 void* **DbRead** (unsigned int * *nRows*, unsigned int * *datetime*, const char * *tableName*, const char * *structName*, unsigned int *nVar*, unsigned int *sizeOfStruct*, **StDbBroker::oldDescriptor** * *d*, const char * *database*, const char * *tableVersion*)

Definition at line 43 of file DbRead.cxx.

```
52 {
53
54
55 char validFrom[20];
56 char validTo[20];
57 char temps[128];
58 char row[128];
59 char currentDateTime[20];
60 char time[7];
61
62 sprintf(currentDateTime,"% .8d",datetime[0]);
63 sprintf(time,"% .6d",datetime[1]);
64 strcat(currentDateTime,time);
65
66 currentDateTime[19]='\0';
67 currentDateTime[18]=currentDateTime[13];
68 currentDateTime[17]=currentDateTime[12];
69 currentDateTime[16]=':';
70 currentDateTime[15]=currentDateTime[11];
71 currentDateTime[14]=currentDateTime[10];
72 currentDateTime[13]=':';
73 currentDateTime[12]=currentDateTime[9];
74 currentDateTime[11]=currentDateTime[8];
75 currentDateTime[10]=' ';
76 currentDateTime[9]=currentDateTime[7];
77 currentDateTime[8]=currentDateTime[6];
78 currentDateTime[7]='-';
79 currentDateTime[6]=currentDateTime[5];
80 currentDateTime[5]=currentDateTime[4];
81 currentDateTime[4]='-';
82
```

```

83 //cout<<"currentDateTime: \"<<currentDateTime<<\"\"<<endl;
84 StDbManager * mgr = StDbManager::Instance();
85
86
87 char dbType[64];
88 char dbDomain[64];
89 StDbType type;
90 StDbDomain domain;
91 char version[128];
92
93 if(!tableVersion)strcpy((char*)version,"default");
94 if(!database){
95     strcpy((char*)dbType,"TestScheme");
96     strcpy((char*)dbDomain,"Star");
97     type = mgr->getDbType(dbType);
98     domain = mgr->getDbDomain(dbDomain);
99 } else {
100     char* atype;
101     char* adomain;
102     if(!mgr->getDataBaseInfo(database, atype, adomain)){
103         cerr << "StDbManager:: DataBase specified incorrectly" << endl;
104         *nRows=0;
105         return NULL;
106     }
107     // cout << "Returned DbType = "<<atype<<" & DbDomain= "<<adomain<< endl;
108     type = mgr->getDbType(atype);
109     domain = mgr->getDbDomain(adomain);
110 }
111 }
112
113
114 StDbConfigNode* node=mgr->initConfig(type,domain);
115 // now try without descriptor --> can't
116 StDbTable* mtable=node->addDbTable(tableName,version);
117 //StDbTable* mtable=node->addTable(tableName,version);
118
119 if(!mtable){
120     *nRows=0;
121     return NULL;
122 }
123
124
125 // --> can't do it 'cause St_base descriptor's limits
126 //_Descriptor* _d = (_Descriptor*)d;
127 //StDbTableDescriptor* dbDescr = new StDbTableDescriptor(_d,nVar,sizeOfStruct);
128 //mtable->setDescriptor((StTableDescriptorI*)dbDescr);
129
130 mgr->setRequestTime(currentDateTime);
131 mgr->fetchDbTable(mtable);
132
133
134 if (mtable->GetTable() ==NULL)
135 {
136     *nRows=0;
137     return NULL;
138 }
139
140 *nRows = mtable->GetNRows();
141
142 // cout<<"DbRead: nRows"<<*nRows<<endl;
143
144 //fill return datetime values
145 int latestDirDate;
146 int latestDirTime;
147 int ic;
148 int i2, i3;
149

```

```
150 //convert hhmmss from: 1999-06-17 12:48:33
151 strcpy(row,mtable->getBeginDateTime());
152
153     strncpy(validFrom,row,19);validFrom[19]='\0';
154     //start from blank at position row[0][10]
155     ic=10;
156     for(i3=0;i3<3;i3++,++ic) {
157         for(i2=0;i2<2;i2++,++ic) {
158             temps[ic]=row[ic];
159         }
160     }
161     temps[6]='\0';
162     latestDirTime = atoi(temps);
163
164     //get date from: 1999-06-17 12:48:33
165
166     strncpy(temps,validFrom,10);
167     temps[4]=temps[5];
168     temps[5]=temps[6];
169     temps[6]=temps[8];
170     temps[7]=temps[9];
171     temps[8]='\0';
172
173     latestDirDate = atoi(temps);
174
175 int nextDirDate;
176 int nextDirTime;
177
178 //convert hhmmss from: 1999-06-17 12:48:33
179 strcpy(row,mtable->getEndDateTime());
180     strncpy(validTo,row,19);validTo[19]='\0';
181     //start from blank at position row[0][10]
182     ic=10;
183     for(i3=0;i3<3;i3++,++ic) {
184         for(i2=0;i2<2;i2++,++ic) {
185             temps[ic]=row[ic];
186         }
187     }
188     temps[6]='\0';
189     nextDirTime = atoi(temps);
190
191     //get date from: 1999-06-17 12:48:33
192     strncpy(temps,validTo,10);
193     temps[4]=temps[5];
194     temps[5]=temps[6];
195     temps[6]=temps[8];
196     temps[7]=temps[9];
197     temps[8]='\0';
198
199     nextDirDate = atoi(temps);
200
201 datetime[0] = latestDirDate;
202 datetime[1] = latestDirTime;
203 datetime[2] = nextDirDate;
204 datetime[3] = nextDirTime;
205
206 if (datetime[2]==19691231) datetime[2]=20380101;
207
208 void* data = mtable->GetTableCpy();
209 delete node;
210
211
212 return data;
213 }
```

5.9 DbUse.hxx File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "Stiostream.h"
#include "Stsstream.h"
#include "mysql.h"
#include "mysql_com.h"
#include "DbEndian.h"
#include "StDbBroker.h"
```

Functions

- void * **DbUse** (uint *nRows, uint *datetime, const char *tableName, const char *structName, uint nVar, uint sizeOfStruct, StDbBroker::oldDescriptor *d)

5.9.1 Function Documentation

5.9.1.1 void* DbUse (uint * nRows, uint * datetime, const char * tableName, const char * structName, uint nVar, uint sizeOfStruct, StDbBroker::oldDescriptor * d)

Definition at line 67 of file DbUse.hxx.

```
74 {
75     enum EColumnType {kNAN, kFloat, kInt, kLong, kShort, kDouble, kUInt
76                 ,kULong, kUShort, kUChar, kChar };
77
78 //cout << "DbUse, event date time " << datetime[0]<< " " << datetime[1] << endl;
79 //    cout << "DbUse, structure: "<<structName<<, table: "<<tableName<< endl;
80 //    cout << " nVar"<<nVar<<, sizeOfStruct " <<sizeOfStruct<< endl;
81
82     uint j;
83     uint count;
84
85     MYSQL mysql;
86     MYSQL_RES *result;
87     MYSQL_ROW row;
88
89     unsigned int num_fields;
90     unsigned int num_rows;
91 //unsigned int num_struct;
92
93 //const int MAXBUF=1024;
94     ostrstream Query;
95     char temps[128];
96
97     char validFrom[20];
98     char currentDateTime[20];
99     char time[7];
100
101    sprintf(currentDateTime,"% .8d",datetime[0]);
102    sprintf(time,"% .6d",datetime[1]);
103    strcat(currentDateTime,time);
```

```

104
105 currentDateTime[19]='\0';
106 currentDateTime[18]=currentDateTime[13];
107 currentDateTime[17]=currentDateTime[12];
108 currentDateTime[16]=':';
109 currentDateTime[15]=currentDateTime[11];
110 currentDateTime[14]=currentDateTime[10];
111 currentDateTime[13]=':';
112 currentDateTime[12]=currentDateTime[9];
113 currentDateTime[11]=currentDateTime[8];
114 currentDateTime[10]=' ';
115 currentDateTime[9]=currentDateTime[7];
116 currentDateTime[8]=currentDateTime[6];
117 currentDateTime[7]='-';
118 currentDateTime[6]=currentDateTime[5];
119 currentDateTime[5]=currentDateTime[4];
120 currentDateTime[4]='-';
121
122 mysql_init(&mysql);
123
124 //set timeout in seconds for bnl.local domain
125
126 //on sun:
127 //Program received signal SIGBUS, Bus error.
128 //0xed737d68 in mysql_options ()
129
130 //mysql_options(&mysql,MYSQL_OPT_CONNECT_TIMEOUT,"4");
131
132 // Establish a connection to the MySQL database engine
133
134 char *dbName=new char[strlen("params")+1]; strcpy(dbName,"params");
135 //only db1 is visible from rcas0202 machine
136 char *dbHost=new char[strlen("db1.star.bnl.gov")+1];strcpy(dbHost,"db1.star.bnl.gov");
137 //char *dbHost="duvall.star.bnl.gov";
138
139 //if (!mysql_real_connect(&mysql,"localhost","","","",dbName,0,"/tmp/mysql.sock",0))
140 //mysql_real_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd, const char
141 //if (!mysql_real_connect(&mysql,"duvall.star.bnl.gov","","","",dbName,0,NULL,0))
142
143 if (!mysql_real_connect(&mysql,dbHost,"","",dbName,0,NULL,0))
144 {
145     cerr << "Failed to connect to database: Error: "
146         << mysql_error(&mysql) << endl;
147     *nRows=0;
148     return NULL;
149 }
150
151 //check if the instance+structure pair already exist
152
153 Query.seekp(0);
154 if(strlen(tableName)>0)
155 {
156     //find latest date
157     Query << "SELECT DISTINCT MAX(validFrom) FROM instances, structures WHERE instances.strI
158 // cout << "database query: " << Query.str() << endl;
159 }
160 else
161 {
162     cerr<<"ERROR: Zero length for Table Name is not allowed"<<endl;
163     return NULL;
164 }
165
166 uint num_latest;
167 int latestDirDate;
168 int latestDirTime;
169
170 if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))

```

```

171  {
172      cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
173      mysql_close(&mysql);
174      return NULL;
175  }
176 else // query succeeded, get result
177  {
178     result = mysql_store_result(&mysql);
179     if (result)
180     {
181         num_fields = mysql_num_fields(result);
182         if (num_fields!=1) cerr << "ERROR: wrong size of LATEST query" << endl;
183
184         num_latest = mysql_num_rows(result);
185
186         if (num_latest==0)// found nothing
187         {
188             cerr << "INFO: db " << dbName << " on host " << dbHost
189             << " has no struct+table pair "<< structName << "+" << tableName
190             << " valid for " << currentDateTime << endl;
191             mysql_close(&mysql);
192             return NULL;
193         }
194     else // this structure name already exists
195     {
196         if (num_latest>1) cerr << "ERROR: found more than one latest date"
197                         << tableName << endl;
198
199 //           cout<<"Found "<<num_latest<<" struct "<<structName<<" for table "<<tableName<<endl;
200 //           row = mysql_fetch_row(result);
201 //convert hhmmss from: 1999-06-17 12:48:33
202 //           strncpy(validFrom,row[0],19);validFrom[19]='\0';
203 //           //start from blank at position row[0][10]
204 //           int ic=10;
205 //           for(int i3=0;i3<3;i3++,++ic) {
206 //               for(int i2=0;i2<2;i2++,++ic) {
207 //                   temps[ic]=row[0][ic];
208 //               }
209 //           }
210 //           temps[6]='\0';
211 //           latestDirTime = atoi(temps);
212
213 //           //get date from: 1999-06-17 12:48:33
214
215 //           strncpy(temps,validFrom,10);
216 //           temps[4]=temps[5];
217 //           temps[5]=temps[6];
218 //           temps[6]=temps[8];
219 //           temps[7]=temps[9];
220 //           temps[8]='\0';
221
222 //           latestDirDate = atoi(temps);
223     }
224     mysql_free_result(result);
225   }
226 else // query OK but, no result?!
227  {
228     cerr << "no result: Error: " << mysql_error(&mysql) << endl;
229     mysql_close(&mysql);
230     return NULL;
231   }
232 }
233
234 //find if there are many entries with the latest date
235 Query.seekp(0);
236     Query << "SELECT DISTINCT instances.ID, instances.nRows, instances.strID, structures.sizeOfStruct",
237

```

```

238 //cout << "database query: " << Query.str() << endl;
239
240 uint num_instances=99999;
241 int latestDirID=99999;
242 int latestStrID=99999;
243 uint sizeOfDbStruct=99999;
244 uint nDbVar=99999;
245
246 if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
247 {
248     cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
249     mysql_close(&mysql);
250     return NULL;
251 }
252 else // query succeeded, get result
253 {
254     result = mysql_store_result(&mysql);
255     if (result)
256     {
257         num_fields = mysql_num_fields(result);
258         if (num_fields!=5) cerr << "ERROR: wrong size of latest entries query" << endl;
259
260         num_instances = mysql_num_rows(result);
261
262         if (num_instances==0)// found nothing
263         {
264             cerr << "ERROR: db " << dbName << " has lost the struct+table pair "
265                 << structName << "+" << tableName << endl;
266             mysql_close(&mysql);
267             return NULL;
268         }
269     else
270     {
271         if (num_instances>1)// bad
272         {
273             cerr << "INFO: db " << dbName << " has more then one struct+table pair "
274                 << structName << "+" << tableName
275                 << " valid for " << currentDateTime
276                 << ", the LAST INSERTED is used" << endl;
277             cerr << "database query: " << Query.str() << endl;
278         }
279         for (uint id=0;id<num_instances;id++)
280         {
281             row = mysql_fetch_row(result);
282             latestDirID = atoi(row[0]);
283             *nRows = (uint) atoi(row[1]);
284             latestStrID = atoi(row[2]);
285             sizeOfDbStruct = (uint)atoi(row[3]);
286             nDbVar = (uint)atoi(row[4]);
287         }
288     }
289     mysql_free_result(result);
290 }
291 else // query OK but, no result?!
292 {
293     cerr << "no result: Error: " << mysql_error(&mysql) << endl;
294     mysql_close(&mysql);
295     return NULL;
296 }
297 }
298
299 Query.seekp(0);
300
301 if(sizeOfDbStruct!=sizeOfStruct)
302 {
303     cerr<<"ERROR: DB struct byteSize is "<<sizeOfDbStruct<<endl;
304     cerr<<"      user struct byteSize is "<<sizeOfStruct<<endl;

```

```

305     cerr << "structure: "<<structName<<endl;
306     *nRows=0;
307     return NULL;
308 }
309
310 if(nVar!=nDbVar)
311 {
312     cerr << "ERROR: DB struct nVariables is "<<nDbVar<<endl;
313     cerr << "      user struct nVariables is "<<nVar<<endl;
314     cerr << "structure: "<<structName<<endl;
315     *nRows = 0;
316     return NULL;
317 }
318
319 //to compare header file info in db and descriptor fetch db header info
320 char **types =0;
321 char **names =0;
322 int *offset=0;
323 int *nDims=0;
324 int *firstDim=0;
325
326
327 Query.seekp(0);
328 Query<<"SELECT name, type, offset, nDims, firstDim FROM headers WHERE strID="
329     << latestStrID << " ORDER BY offset" << endl;
330 //cout << "database query: " << Query.str() << endl;
331
332 if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
333 {
334     cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
335     mysql_close(&mysql);
336     return NULL;
337 }
338 else // query succeeded, get result
339 {
340     result = mysql_store_result(&mysql);
341     if (result)// query OK
342     {
343         num_fields = mysql_num_fields(result);
344         if (num_fields!=5) cerr<<"ERROR: wrong size of headers query"<<endl;
345
346         num_rows = mysql_num_rows(result);
347         //           cout<<"total variables: "<< num_rows << endl;
348
349         if (num_rows!=nVar) //size is different from this ID
350         {
351             cout<<"WARNING: database structure is of "<< num_rows
352                 <<" variables, this structure is "<< nVar << endl;
353         }
354
355         types   = new char*[num_rows];
356         names   = new char*[num_rows];
357         offset   = new int[num_rows];
358         nDims   = new int[num_rows];
359         firstDim = new int[num_rows];
360
361         for (j=0;j<num_rows;j++)
362         {
363             row = mysql_fetch_row(result);
364
365             names[j] = strdup(row[0]);
366             types[j] = strdup(row[1]);
367             offset[j] = atoi(row[2]);
368             nDims[j] = atoi(row[3]);
369             firstDim[j] = atoi(row[4]);
370
371             //cout<<"name: \\"<<row[0]<<"\\"<<endl;

```

```

372         //cout<<"type: \\"<<row[1]<<"\\"<<endl;
373         //cout<<"offset: "<<offset[j]<<endl;
374         //cout<<"nDims: "<<nDims[j]<<endl;
375         //cout<<"firstDim: "<<firstDim[j]<<endl;
376     }
377     mysql_free_result(result);
378 }
379 }
380
381 //now compare header file info in db and descriptor
382
383 //same variables counter
384 count = 0;
385
386 for (j=0;j<nVar;j++)
387 {
388     if (strcmp(d[j].fColumnName,names[j]))
389         break;
390
391     //cout<<"offset: \\"<<d[j].offset<<"\\"<<row[1]<<"\\"<<endl;
392     if ( d[j].fOffset!=(unsigned int)offset[j] )
393         break;
394
395     //same offset: check nDims
396     //cout<<"nDims: \\"<<d[j].dimensions<<"\\"<<row[2]<<"\\"<<endl;
397     if ( d[j].fDimensions!=(unsigned int)nDims[j] )
398         break;
399
400     //same nDims: check firstDim
401     //cout<<"nDims: \\"<<d[j].firstDimension<<"\\"<<row[3]<<"\\"<<endl;
402     if ( d[j].fIndexArray[0]!=(unsigned int)firstDim[j] )
403         break;
404
405     //cout<<"variable "<< j+1 <<" is the same"<<endl;
406     count++;
407 }
408
409 if (count!=nVar)
410 {
411     cerr<<"structure " << structName
412     <<" is not compatible with database instance ID "<<latestDirID<<endl;
413     cerr <<nVar <<nVar<<, count <<count<<endl;
414
415     for (j=0;j<nVar;j++)
416     {
417         cerr<<"names: \\"<<d[j].fColumnName<<"\\"<<names[j]<<"\\"<<endl;
418         if (strcmp(d[j].fColumnName,names[j]))
419             break;
420
421         cerr<<"offset: "<<d[j].fOffset<<"\\"<<offset[j]<<endl;
422         if ( d[j].fOffset!=(unsigned int)offset[j] )
423             break;
424
425         cerr<<"nDims: "<<d[j].fDimensions<<"\\"<<nDims[j]<<endl;
426         if ( d[j].fDimensions!=(unsigned int)nDims[j] )
427             break;
428
429         cerr<<"firstDim: "<<d[j].fIndexArray[0]<<"\\"<<firstDim[j]<<endl;
430         if ( d[j].fIndexArray[0]!=(unsigned int)firstDim[j] )
431             break;
432     }
433
434     *nRows=0;
435     return NULL;
436 }
437
438 //char* pCurrent=new char[num_struct*sizeOfStruct];

```

```

439 //void * pDbData = (void*) pCurrent;
440 //Valeri Fine need's malloc() for ROOT free()
441 //void* pCurrent = malloc((size_t) num_struct*sizeOfStruct);
442 //use calloc() for array of structs?
443 void* pDbData = calloc((size_t) *nRows, (size_t) sizeOfDbStruct);
444
445 if (pDbData==NULL){
446     cerr<<"DbUse: failed to allocate memory needed for the array of "
447         << *nRows << " structs " <<structName << " each of "
448         <<sizeOfStruct << " bytes"<<endl;
449     *nRows=0;
450     return NULL;
451 }
452
453 char* pCurrent = (char*) pDbData;
454
455 uint num_blobs;
456 unsigned long *lengths;
457
458 Query.seekp(0);
459 Query << "SELECT bytes FROM bytes WHERE instanceID="<<latestDirID<< endl;
460
461 // cout << "database query: " << Query.str() << endl;
462
463 if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
464 {
465     cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
466     cerr << "database query: " << Query.str() << endl;
467     mysql_close(&mysql);
468     return NULL;
469 }
470 else // query succeeded, get result
471 {
472     result = mysql_store_result(&mysql);
473     if (!result) // query OK but, no result?!
474     {
475         cerr << "no result: Error: " << mysql_error(&mysql) << endl;
476         mysql_close(&mysql);
477         return NULL;
478     }
479     else
480     {
481         num_fields = mysql_num_fields(result);
482         if (num_fields!=1) cerr << "ERROR: wrong size of blob query"<<endl;
483
484         num_blobs = mysql_num_rows(result);
485
486         if (num_blobs==0)// found nothing
487         {
488             cerr << "ERROR: db " << dbName << " has lost BLOB for struct+table pair "
489                 <<structName<< "+"<< tableName
490                 << " valid for "<<currentTime <<endl;
491             mysql_close(&mysql);
492             return NULL;
493         }
494     else
495     {
496         if (num_blobs>1) cerr << "ERROR: found more than one BLOB for "
497                         <<tableName<<endl;
498         //this is the place where data are fethched
499         row = mysql_fetch_row(result);
500
501         if (row)
502         {
503             lengths = mysql_fetch_lengths(result);
504
505             //cout << "blob size "<<lengths[0]<< " for table "<<tableName<<endl;

```

```

506             if (lengths[0] != (*nRows)*sizeOfDbStruct)
507             {
508                 cerr << "ERROR: wrong blob size "
509                     << tableName << endl;
510                 cerr << "lengths[0] " << lengths[0]
511                     << ", nRows " << (*nRows)
512                     << ", sizeOfDbStruct " << sizeOfDbStruct
513                     << ", nRows*sizeOfDbStruct " << (*nRows)*sizeOfDbStruct
514                     << endl;
515                 mysql_close(&mysql);
516                 *nRows=0;
517                 return NULL;
518             }
519
520 #ifndef BIG_ENDIAN
521         memcpy(pCurrent, row[0], (size_t) (*nRows)*sizeOfDbStruct);
522 #else // do byte swapping
523         //memset(pCurrent, 0, (size_t) (*nRows)*sizeOfDbStruct);
524         uint i, k, nTimes, firstByte, firstDbByte;
525         //convert to user type (not Db type)
526
527         for (i=0;i<*nRows;i++)
528         {
529             for (j=0;j<nVar;j++)
530             {
531
532                 switch(d[j].fDimensions)
533                 {
534                     case 0:
535                         nTimes=1;
536                         break;
537                     case 1:
538                         nTimes=d[j].fIndexArray[0];
539                         break;
540                     case 2:
541                         cerr << "two-dims not handled yet" << endl;
542                         nTimes=0;
543                         //nTimes=d[j].firstDimension*d[j].secondDimension;
544                         break;
545                     default:
546                         cerr << "ERROR: more than one dimension " << endl;
547                         nTimes=0;
548                         break;
549                 }
550
551                 //from db offset to user offset
552                 firstByte=i*sizeOfStruct+d[j].fOffset;
553                 firstDbByte=i*sizeOfDbStruct+offset[j];
554
555                 for (k=0;k<nTimes;k++) {
556
557                     //MYSQL_ROW row is just char**
558                     switch((StDbBroker::EColumnType)d[j].fType)
559                     {
560                         case kUChar:
561                         case kChar:
562                             pCurrent[firstByte+k]=row[0][firstDbByte+k];
563                             break;
564
565                         case kShort:
566                         case kUShort:
567                             pCurrent[firstByte+2*k+1]=row[0][+2*k];
568                             pCurrent[firstByte+2*k] =row[0][firstDbByte+2*k+1];
569                             break;
570
571                         case kInt:
572                         case kUInt:

```

```

573
574         case kULong:
575         case kLong:
576
577         case kFloat:
578
579             pCurrent[firstByte+4*k+3]=row[0][firstDbByte+4*k];
580             pCurrent[firstByte+4*k+2]=row[0][firstDbByte+4*k+1];
581             pCurrent[firstByte+4*k+1]=row[0][firstDbByte+4*k+2];
582             pCurrent[firstByte+4*k]=row[0][firstDbByte+4*k+3];
583             break;
584
585         case kDouble:
586             //and long long
587             pCurrent[firstByte+8*k+7]=row[0][firstDbByte+8*k];
588             pCurrent[firstByte+8*k+6]=row[0][firstDbByte+8*k+1];
589             pCurrent[firstByte+8*k+5]=row[0][firstDbByte+8*k+2];
590             pCurrent[firstByte+8*k+4]=row[0][firstDbByte+8*k+3];
591             pCurrent[firstByte+8*k+3]=row[0][firstDbByte+8*k+4];
592             pCurrent[firstByte+8*k+2]=row[0][firstDbByte+8*k+5];
593             pCurrent[firstByte+8*k+1]=row[0][firstDbByte+8*k+6];
594             pCurrent[firstByte+8*k]=row[0][firstDbByte+8*k+7];
595             break;
596
597         case kNaN:
598     default:
599         cerr << "ERROR: unknown type!" << endl;
600         break;
601     }
602     } //end loop over array variable
603 } //end loop over variables
604 } //end loop over nRows
605 #endif
606     }
607 }
608 mysql_free_result(result);
609 }
610 }
611
612 // select just one table with the smallest date after the datetime
613 Query.seekp(0);
614     //find latest date
615     Query << "SELECT DISTINCT MIN(instances.validFrom) FROM instances, structures WHERE instances.strI
616
617 // cout << "database query: " << Query.str() << endl;
618
619 uint num_next;
620 int nextDirDate;
621 int nextDirTime;
622
623 if (mysql_real_query(&mysql,Query.str(),Query.pcount()-1))
624 {
625     cerr << "Failed to query: Error: " << mysql_error(&mysql) << endl;
626     mysql_close(&mysql);
627     return NULL;
628 }
629 else // query succeeded, get result
630 {
631     result = mysql_store_result(&mysql);
632     if (result)
633     {
634         num_fields = mysql_num_fields(result);
635         if (num_fields!=1) cerr << "ERROR: wrong size of next date query" << endl;
636
637         num_next = mysql_num_rows(result);
638
639         if (num_next==0)// found nothing no next table = no validity limit

```

```

640      {
641 //           cout << "INFO: db " << dbName << " has no struct+table pair "
642 //               << structName<< "+"<< tableName
643 //               << " limiting validity for "<<setw(8)<<datetime[0]<<setw(6)<<datetime[1]<<endl;
644           nextDirDate=20380101;
645           nextDirTime=0;
646       }
647     else //validity limit exists
648     {
649         if (num_next>1) cerr << "ERROR: found more than one next date"
650                     <<tableName<<endl;
651 //         cout<<"Found next "<<num_next<<" struct "<<structName<<" for table "<<tableName<<endl;
652
653         row = mysql_fetch_row(result);
654
655
656 //convert hhmmss from: 1999-06-17 12:48:33
657         strncpy(validFrom,row[0],19);validFrom[19]='\0';
658         //start from blank at position row[0][10]
659         int ic=10;
660         for(int i3=0;i3<3;i3++,++ic) {
661             for(int i2=0;i2<2;i2++,++ic) {
662                 temps[ic]=row[0][ic];
663             }
664         }
665         temps[6]='\0';
666         nextDirTime = atoi(temps);
667
668         //get date from: 1999-06-17 12:48:33
669         strncpy(temp,validFrom,10);
670         temp[4]=temp[5];
671         temp[5]=temp[6];
672         temp[6]=temp[8];
673         temp[7]=temp[9];
674         temp[8]='\0';
675
676         nextDirDate = atoi(temp);
677     }
678     mysql_free_result(result);
679   }
680 else // query OK but, no result?!
681   {
682     cerr << "no result: Error: " << mysql_error(&mysql) << endl;
683     mysql_close(&mysql);
684     return NULL;
685   }
686 }
687
688 // We are done with the connection, call mysql_close() to terminate it
689
690 mysql_close(&mysql);
691
692 delete [] types;
693 delete [] names;
694 delete [] offset;
695 delete [] nDims;
696 delete [] firstDim;
697
698 //fill return datetime values
699
700 datetime[0] = latestDirDate;
701 datetime[1] = latestDirTime;
702 datetime[2] = nextDirDate;
703 datetime[3] = nextDirTime;
704
705 // cout<<"Begin "<<datetime[0]<<" "<<datetime[1]<<endl;
706 // cout<<"End     "<<datetime[2]<<" "<<datetime[3]<<endl;

```

```
707  
708 //cout <<"nRows = "<<*nRows<<endl;  
709 return pDbData;  
710  
711 }
```

5.10 StDbBroker.cxx File Reference

```
#include <StString.h>
#include <Stiostream.h>
#include <stdlib.h>
#include "TString.h"
#include "TROOT.h"
#include "TBuffer.h"
#include "TClass.h"
#include "St_Table.h"
#include "TRealData.h"
#include "TDataMember.h"
#include "TDataType.h"
#include "Api.h"
#include "StDbBroker.h"
#include "StDbLib/StDbManager.hh"
#include "StDbLib/StDbConfigNode.hh"
#include "StDbLib/StDbTable.h"
#include "dbNodes.h"
#include "StDbLib/StDbTableIter.hh"
#include "StDbLib/StDbBuffer.h"
#include "StDbLib/StDbTableDescriptor.h"
#include "StDbWrappedMessenger.hh"
```

Defines

- #define **_CLASS_** "StDbBroker"
- #define **_METHOD_** "WriteToDb(pArray,tabID)"
- #define **_METHOD_** "WriteToDb(pArray,fullPath,idList)"

Functions

- [ClassImp \(StDbBroker\) char **StDbBroker](#)

5.10.1 Define Documentation

5.10.1.1 #define **_CLASS_** "StDbBroker"

Definition at line 196 of file StDbBroker.cxx.

5.10.1.2 #define __METHOD__ "WriteToDb(pArray fullPath,idList)"

5.10.1.3 #define __METHOD__ "WriteToDb(pArray,tabID)"

5.10.2 Function Documentation

5.10.2.1 ClassImp ([StDbBroker](#))

Definition at line 199 of file StDbBroker.cxx.

```
203 {
204     char **ElementComment = new char*[m_nElements];
205     if (!parentTable) {
206         //    MakeZombie();
207         return NULL;
208     }
209
210     TClass *classPtr = parentTable->GetRowClass();
211     if (!classPtr) return NULL;
212
213     if (!classPtr->GetListOfRealData()) classPtr->BuildRealData();
214
215     TIIter next(classPtr->GetListOfDataMembers());
216     TDataMember *member = 0;
217     UInt_t i=0, j=0;
218     while ( (member = (TDataMember *) next()) ) {
219         ElementComment[i] = strdup(member->GetTitle());
220         // strip trailing blanks from Comments (they are stripped in mysql anyway)
221         j=strlen(ElementComment[i]);
222         while (j != 0 && ElementComment[i][j-1] == ' ') j--;
223         ElementComment[i][j]='\0';
224         i++;
225     }
226     return ElementComment;
227 }
```

5.11 StDbBroker.h File Reference

```
#include "Rtypes.h"
#include "dbNodeArray.h"
#include "dbConfig.h"
#include "St_tableDescriptor.h"
```

Classes

- class [StDbBroker](#)
- struct [StDbBroker::oldDescriptor](#)

Functions

- void [DbFill](#) (unsigned int *, const char *, const char *, unsigned int, [StDbBroker::oldDescriptor](#) *d, const char **, unsigned int, unsigned int, void *)
- void * [DbUse](#) (unsigned int *, unsigned int *, const char *, const char *, unsigned int, unsigned int, [StDbBroker::oldDescriptor](#) *d)
- void * [DbRead](#) (unsigned int *, unsigned int *, const char *, const char *, unsigned int, unsigned int, [StDbBroker::oldDescriptor](#) *d, const char *, const char *)
- int [DbInit](#) (const char *)

5.11.1 Function Documentation

5.11.1.1 void [DbFill](#) (unsigned int *, const char *, const char *, unsigned int, [StDbBroker::oldDescriptor](#) * d, const char **, unsigned int, unsigned int, void *)

5.11.1.2 int [DbInit](#) (const char *)

Definition at line 26 of file DbInit.cxx.

```
27 {
28
29 MySQL mysql;
30
31 mysql_init(&mysql);
32
33 //set timeout in seconds for bnl.local domain
34
35 #ifndef __sun
36 //on sun:
37 //Program received signal SIGBUS, Bus error.
38 //0xed737d68 in mysql_options ()
39
40 mysql_options(&mysql,MYSQL_OPT_CONNECT_TIMEOUT,"2");
41
42 #endif
43 // Try to establish a connection to the MySQL database engine
44
45 const char *database=dbName;
46 //only dbl is visible from rcas0202 machine
47 const char *dbHost="dbl.star.bnl.gov";
48 //char *dbHost="duvall.star.bnl.gov";
49 //mysql_real_connect(MYSQL *mysql, const char *host, const char *user, const char *passwd, const char *
```

```

50
51 if (!mysql_real_connect(&mysql, dbHost, "", "", database, 0, NULL, 0))
52 {
53     cerr << "Failed to connect to database: " << database << endl;
54     cerr << "MySQL ERROR: " << mysql_error(&mysql) << endl;
55     return 1;
56 }
57
58 // We are done with the connection, call mysql_close() to terminate it
59 mysql_close(&mysql);
60
61 return 0;
62
63 }

```

5.11.1.3 void* DbRead (unsigned int *, unsigned int *, const char *, const char *, unsigned int, unsigned int, StDbBroker::oldDescriptor * d, const char *, const char *)

Definition at line 43 of file DbRead.cxx.

```

52 {
53
54
55 char validFrom[20];
56 char validTo[20];
57 char temps[128];
58 char row[128];
59 char currentDateTime[20];
60 char time[7];
61
62 sprintf(currentDateTime, "%.8d", datetime[0]);
63 sprintf(time, "%.6d", datetime[1]);
64 strcat(currentDateTime, time);
65
66 currentDateTime[19] = '\0';
67 currentDateTime[18] = currentDateTime[13];
68 currentDateTime[17] = currentDateTime[12];
69 currentDateTime[16] = ':';
70 currentDateTime[15] = currentDateTime[11];
71 currentDateTime[14] = currentDateTime[10];
72 currentDateTime[13] = ':';
73 currentDateTime[12] = currentDateTime[9];
74 currentDateTime[11] = currentDateTime[8];
75 currentDateTime[10] = ' ';
76 currentDateTime[9] = currentDateTime[7];
77 currentDateTime[8] = currentDateTime[6];
78 currentDateTime[7] = '-';
79 currentDateTime[6] = currentDateTime[5];
80 currentDateTime[5] = currentDateTime[4];
81 currentDateTime[4] = '-';
82
83 //cout << "currentDateTime: " << currentDateTime << endl;
84 StDbManager * mgr = StDbManager::Instance();
85
86
87 char dbType[64];
88 char dbDomain[64];
89 StDbType type;
90 StDbDomain domain;
91 char version[128];
92
93 if (!tableVersion) strcpy((char*)version, "default");
94 if (!database){
95     strcpy((char*)dbType, "TestScheme");

```

```

96     strcpy((char*)dbDomain,"Star");
97     type = mgr->getDbType(dbType);
98     domain = mgr->getDbDomain(dbDomain);
99 } else {
100     char* atype;
101     char* adomain;
102     if(!mgr->get DataBaseInfo(database, atype, adomain)){
103         cerr << "StDbManager:: Database specified incorrectly" << endl;
104         *nRows=0;
105         return NULL;
106     }
107     // cout << "Returned DbType = "<<atype<<" & DbDomain= "<<adomain<< endl;
108     type = mgr->getDbType(atype);
109     domain = mgr->getDbDomain(adomain);
110
111 }
112
113
114 StDbConfigNode* node=mgr->initConfig(type,domain);
115 // now try without descriptor --> can't
116 StDbTable* mtable=node->addDbTable(tableName,version);
117 //StDbTable* mtable=node->addTable(tableName,version);
118
119 if(!mtable){
120     *nRows=0;
121     return NULL;
122 }
123
124
125 // --> can't do it 'cause St_base descriptor's limits
126 //_Descriptor* _d = (_Descriptor*)d;
127 //StDbTableDescriptor* dbDescr = new StDbTableDescriptor(_d,nVar,sizeOfStruct);
128 //mtable->setDescriptor((StTableDescriptorI*)dbDescr);
129
130 mgr->setRequestTime(currentDateTime);
131 mgr->fetchDbTable(mtable);
132
133
134 if (mtable->GetTable()==NULL)
135 {
136     *nRows=0;
137     return NULL;
138 }
139
140 *nRows = mtable->GetNRows();
141
142 // cout<<"DbRead: nRows"<<*nRows<<endl;
143
144 //fill return datetime values
145 int latestDirDate;
146 int latestDirTime;
147 int ic;
148 int i2, i3;
149
150 //convert hhmmss from: 1999-06-17 12:48:33
151 strcpy(row,mtable->getBeginDateTime());
152
153         strncpy(validFrom,row,19);validFrom[19]='\0';
154         //start from blank at position row[0][10]
155         ic=10;
156         for(i3=0;i3<3;i3++,++ic) {
157             for(i2=0;i2<2;i2++,++ic) {
158                 temps[ic]=row[ic];
159             }
160         }
161         temps[6]='\0';
162         latestDirTime = atoi(temps);

```

```

163
164         //get date from: 1999-06-17 12:48:33
165
166         strncpy(temp,validFrom,10);
167         temps[4]=temps[5];
168         temps[5]=temps[6];
169         temps[6]=temps[8];
170         temps[7]=temps[9];
171         temps[8]='\0';
172
173         latestDirDate = atoi(temp);
174
175 int nextDirDate;
176 int nextDirTime;
177
178 //convert hhmmss from: 1999-06-17 12:48:33
179 strcpy(row,mtable->getEndDateTime());
180         strncpy(validTo,row,19);validTo[19]='\0';
181         //start from blank at position row[0][10]
182         ic=10;
183         for(i3=0;i3<3;i3++,++ic) {
184             for(i2=0;i2<2;i2++,++ic) {
185                 temps[ic]=row[ic];
186             }
187         }
188         temps[6]='\0';
189         nextDirTime = atoi(temp);
190
191         //get date from: 1999-06-17 12:48:33
192         strncpy(temp,validTo,10);
193         temps[4]=temps[5];
194         temps[5]=temps[6];
195         temps[6]=temps[8];
196         temps[7]=temps[9];
197         temps[8]='\0';
198
199         nextDirDate = atoi(temp);
200
201 datetime[0] = latestDirDate;
202 datetime[1] = latestDirTime;
203 datetime[2] = nextDirDate;
204 datetime[3] = nextDirTime;
205
206 if (datetime[2]==19691231) datetime[2]=20380101;
207
208 void* data = mtable->GetTableCpy();
209 delete node;
210
211
212 return data;
213 }
```

5.11.1.4 void* DbUse (unsigned int *, unsigned int *, const char *, const char *, unsigned int, unsigned int, StDbBroker::oldDescriptor * d)

5.12 StDbBrokerLinkDef.h File Reference

5.13 StDbWrappedMessenger.cc File Reference

```
#include "StDbWrappedMessenger.hh"
#include <string.h>
#include "StUtilities/StMessageManager.h"
```

5.14 StDbWrappedMessenger.hh File Reference

```
#include "StDbLib/StDbMessService.hh"
```

Classes

- class [StDbWrappedMessenger](#)

Index

~StDbBroker
 StDbBroker, 18
~StDbWrappedMessenger
 StDbWrappedMessenger, 39
~dbNodeArray
 dbNodeArray, 9
~dbNodes
 dbNodes, 11
__CLASS__
 StDbBroker.cxx, 67
__METHOD__
 StDbBroker.cxx, 67, 68

addNode
 dbNodeArray, 9
 dbNodes, 12

buildConfig
 StDbBroker, 18
buildNodes
 StDbBroker, 19

ClassImp
 StDbBroker.cxx, 68
CloseAllConnections
 StDbBroker, 20
curNode
 dbNodes, 14

dbConfig.h, 41
dbConfig_st, 7
dbConfig_st
 parID, 7
 parname, 7
 tabID, 7
 tablename, 7
 tabtype, 7
DbEndian.h, 42
DbFill
 DbFill.cxx, 43
 StDbBroker.h, 69
DbFill.cxx, 43
DbFill.cxx
 DbFill, 43
 strmov, 48
DbInit
 DbInit.cxx, 49
 StDbBroker, 20
 StDbBroker.h, 69
 DbInit.cxx, 49
 DbInit.cxx
 DbInit, 49
 dbNodeArray, 9
 dbNodeArray
 ~dbNodeArray, 9
 addNode, 9
 getNode, 9
 getNumNodes, 9
 getParent, 9
 getParentID, 9
 next, 10
 reset, 10
 dbNodeArray.h, 50
 dbNodes, 11
 dbNodes, 11
 dbNodes
 ~dbNodes, 11
 addNode, 12
 curNode, 14
 dbNodes, 11
 deleteLists, 12
 extendParentList, 12
 getNode, 12
 getNumNodes, 13
 getParent, 13
 getParentID, 13
 maxList, 14
 mnodes, 14
 mpids, 14
 next, 13
 numNodes, 14
 reset, 14
 dbNodes.cc, 51
 dbNodes.h, 52
 dbNodes.h
 nodeVec, 52
 DbRead
 DbRead.cxx, 53
 StDbBroker.h, 70
 DbRead.cxx, 53
 DbRead.cxx

DbRead, 53
DbUse
 DbUse.hxx, 56
 StDbBroker.h, 72
DbUse.hxx, 56
DbUse.hxx
 DbUse, 56
deleteLists
 dbNodes, 12
Descriptor
 StDbBroker, 17

EColumnType
 StDbBroker, 17
extendParentList
 dbNodes, 12

fColumnName
 StDbBroker::oldDescriptor, 38
fDimensions
 StDbBroker::oldDescriptor, 38
Fill
 StDbBroker, 20
fIndexArray
 StDbBroker::oldDescriptor, 38
findTable
 StDbBroker, 21
fOffset
 StDbBroker::oldDescriptor, 38
fSize
 StDbBroker::oldDescriptor, 38
fType
 StDbBroker::oldDescriptor, 38
fTypeSize
 StDbBroker::oldDescriptor, 38

GetBeginDate
 StDbBroker, 21
GetBeginTime
 StDbBroker, 21
GetBeginTimeStamp
 StDbBroker, 21
GetComments
 StDbBroker, 21
GetEndDate
 StDbBroker, 22
GetEndTime
 StDbBroker, 22
GetEndTimeStamp
 StDbBroker, 22
GetFlavor
 StDbBroker, 22
getNode
 dbNodeArray, 9
 dbNodes, 12
 GetNRows
 StDbBroker, 22
 getNumNodes
 dbNodeArray, 9
 dbNodes, 13
 getParent
 dbNodeArray, 9
 dbNodes, 13
 getParentID
 dbNodeArray, 9
 dbNodes, 13
 GetProdTime
 StDbBroker, 22
 GetRequestTimeStamp
 StDbBroker, 22
 GetTableDescriptor
 StDbBroker, 22
 GetTypeName
 StDbBroker, 24

InitConfig
 StDbBroker, 24
IsZombie
 StDbBroker, 25

kChar
 StDbBroker, 17
kDouble
 StDbBroker, 17
kFloat
 StDbBroker, 17
kInt
 StDbBroker, 17
kLong
 StDbBroker, 17
kNAN
 StDbBroker, 17
kShort
 StDbBroker, 17
kUChar
 StDbBroker, 17
kUInt
 StDbBroker, 17
kULong
 StDbBroker, 17
kUShort
 StDbBroker, 17
loadOldDescriptor
 StDbBroker, 25
m_BeginDate
 StDbBroker, 34

m_BeginTime
 StDbBroker, 34
 m_beginTimeStamp
 StDbBroker, 34
 m_database
 StDbBroker, 34
 m_DateTime
 StDbBroker, 34
 m_descriptor
 StDbBroker, 34
 m_EndDate
 StDbBroker, 35
 m_EndTime
 StDbBroker, 35
 m_endTimeStamp
 StDbBroker, 35
 m_flavor
 StDbBroker, 35
 m_isVerbose
 StDbBroker, 35
 m_isZombie
 StDbBroker, 35
 m_nElements
 StDbBroker, 35
 m_node
 StDbBroker, 35
 m_Nodes
 StDbBroker, 35
 m_nRows
 StDbBroker, 35
 m_ParentType
 StDbBroker, 35
 m_prodTime
 StDbBroker, 36
 m_requestTimeStamp
 StDbBroker, 36
 m_runNumber
 StDbBroker, 36
 m_sizeOfStruct
 StDbBroker, 36
 m_structName
 StDbBroker, 36
 m_tableName
 StDbBroker, 36
 m_tableVersion
 StDbBroker, 36
 m_Tree
 StDbBroker, 36
 makeDateTime
 StDbBroker, 26
 maxList
 dbNodes, 14
 mdescriptor
 StDbBroker, 36

mgr
 StDbBroker, 36
 mMessenger
 StDbWrappedMessenger, 40
 mnodes
 dbNodes, 14
 mpids
 dbNodes, 14

next
 dbNodeArray, 10
 dbNodes, 13
 nodeVec
 dbNodes.h, 52
 numNodes
 dbNodes, 14

parID
 dbConfig_st, 7
 parname
 dbConfig_st, 7
 printMessage
 StDbWrappedMessenger, 39, 40
 printStatistics
 StDbBroker, 26

reset
 dbNodeArray, 10
 dbNodes, 14

SetBeginDate
 StDbBroker, 26
 SetBeginTime
 StDbBroker, 26
 SetBeginTimeStamp
 StDbBroker, 26
 SetDataBaseName
 StDbBroker, 26
 SetDateTime
 StDbBroker, 27
 SetDictionary
 StDbBroker, 27
 SetEndDate
 StDbBroker, 27
 SetEndTime
 StDbBroker, 27
 SetEndTimeStamp
 StDbBroker, 28
 SetFlavor
 StDbBroker, 28
 SetNRows
 StDbBroker, 28
 SetProdTime
 StDbBroker, 28

SetRequestTimeStamp
 StDbBroker, 28
SetRunNumber
 StDbBroker, 28
SetStructName
 StDbBroker, 29
SetStructSize
 StDbBroker, 29
SetTableFlavor
 StDbBroker, 29
SetTableName
 StDbBroker, 29
setVerbose
 StDbBroker, 29
SetVersionName
 StDbBroker, 29
SetZombie
 StDbBroker, 30
StDbBroker, 15
 kChar, 17
 kDouble, 17
 kFloat, 17
 kInt, 17
 kLong, 17
 kNAN, 17
 kShort, 17
 kUChar, 17
 kUInt, 17
 kULong, 17
 kUShort, 17
 StDbBroker, 17
StDbBroker
 ~StDbBroker, 18
 buildConfig, 18
 buildNodes, 19
 CloseAllConnections, 20
 DbInit, 20
 Descriptor, 17
 EColumnType, 17
 Fill, 20
 findTable, 21
 GetBeginDate, 21
 GetBeginTime, 21
 GetBeginTimeStamp, 21
 GetComments, 21
 GetEndDate, 22
 GetEndTime, 22
 GetEndTimeStamp, 22
 GetFlavor, 22
 GetNRows, 22
 GetProdTime, 22
 GetRequestTimeStamp, 22
 GetTableDescriptor, 22
 GetTypeNames, 24
 InitConfig, 24
 IsZombie, 25
 loadOldDescriptor, 25
 m_BeginDate, 34
 m_BeginTime, 34
 m_beginTimeStamp, 34
 m_database, 34
 m_DateTime, 34
 m_descriptor, 34
 m_EndDate, 35
 m_EndTime, 35
 m_endTimeStamp, 35
 m_flavor, 35
 m_isVerbose, 35
 m_isZombie, 35
 m_nElements, 35
 m_node, 35
 m_Nodes, 35
 m_nRows, 35
 m_ParentType, 35
 m_prodTime, 36
 m_requestTimeStamp, 36
 m_runNumber, 36
 m_sizeOfStruct, 36
 m_structName, 36
 m_tableName, 36
 m_tableVersion, 36
 m_Tree, 36
 makeDateTime, 26
 mdescriptor, 36
 mgr, 36
 printStatistics, 26
 SetBeginDate, 26
 SetBeginTime, 26
 SetBeginTimeStamp, 26
 SetDataBaseName, 26
 SetDateTime, 27
 SetDictionary, 27
 SetEndDate, 27
 SetEndTime, 27
 SetEndTimeStamp, 28
 SetFlavor, 28
 SetNRows, 28
 SetProdTime, 28
 SetRequestTimeStamp, 28
 SetRunNumber, 28
 SetStructName, 29
 SetStructSize, 29
 SetTableFlavor, 29
 SetTableName, 29
 setVerbose, 29
 SetVersionName, 29
 SetZombie, 30
 StDbBroker, 17

Use, 30, 31
UseRunLog, 32
WriteToDb, 32, 33
StDbBroker.hxx, 67
StDbBroker.hxx
 __CLASS__, 67
 __METHOD__, 67, 68
 ClassImp, 68
StDbBroker.h, 69
StDbBroker.h
 DbFill, 69
 DbInit, 69
 DbRead, 70
 DbUse, 72
StDbBroker::oldDescriptor, 38
StDbBroker::oldDescriptor
 fColumnName, 38
 fDimensions, 38
 fIndexArray, 38
 fOffset, 38
 fSize, 38
 fType, 38
 fTypeSize, 38
StDbBrokerLinkDef.h, 73
StDbWrappedMessenger, 39
 StDbWrappedMessenger, 39
StDbWrappedMessenger
 ~StDbWrappedMessenger, 39
 mMessenger, 40
 printMessage, 39, 40
 StDbWrappedMessenger, 39
StDbWrappedMessenger.cc, 74
StDbWrappedMessenger.hh, 75
strmov
 DbFill.hxx, 48

tabID
 dbConfig_st, 7
tablename
 dbConfig_st, 7
tabtype
 dbConfig_st, 7

Use
 StDbBroker, 30, 31
UseRunLog
 StDbBroker, 32

WriteToDb
 StDbBroker, 32, 33